LabVIEW™

System Identification Toolkit User Manual

August 2006
371001C-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 6555 7838, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, India 91 80 41190000, Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Lebanon 961 0 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 1800 226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 02 2377 2222, Thailand 662 278 6777, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

© 2004–2006 National Instruments Corporation. All rights reserved.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on `ni.com/legal` for more information about National Instruments trademarks.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

Conventions ix
Related Documentation x

Chapter 1
Introduction to System Identification

Model-Based Control Design Process 1-2
Analyzing Data and Creating a Dynamic System Model 1-2
Designing a Controller 1-4
Simulating the Dynamic System 1-4
Deploying the Controller 1-4
Model Types and Model Representations 1-5
Model Types 1-5
Linear versus Nonlinear Models 1-5
Time-Variant versus Time-Invariant Models 1-5
Continuous versus Discrete Models 1-6
Model Representations 1-6
Developing Models with the System Identification Toolkit 1-7
System Identification VIs 1-7
System Identification Assistant 1-7
System Identification Procedure 1-8

Chapter 2
Acquiring and Preprocessing Data

Acquiring Data from a System 2-1
Accounting for Factors that Influence a System 2-2
Choosing a Stimulus Signal 2-2
Common Stimulus Signals 2-3
Filtered Gaussian White Noise 2-4
Random Binary Signal 2-4
Pseudo-Random Binary Sequence 2-5
Chirp Waveform 2-6
Selecting a Sampling Rate 2-7
Applying an Anti-Aliasing Filter 2-7
Preprocessing Data from a System 2-8
Visually Inspecting the Data 2-8

Removing Offsets and Trends 2-8
Removing Offsets 2-9
Removing Trends 2-9
Outliers 2-10
Filtering and Downsampling 2-10
Data Scaling 2-11

Chapter 3
Nonparametric Model Estimation Methods

Impulse Response 3-2
Least Squares 3-3
Correlation Analysis 3-4
Prewhitening 3-5
Accuracy of the Impulse Response 3-6
Selecting Impulse Response Length 3-9
Applications of the Impulse Response 3-9
Frequency Response 3-12
Spectral Analysis Method 3-12
Accuracy of the Lag Window 3-14
Applications of the Frequency Response 3-17

Chapter 4
Parametric Model Estimation Methods

Parametric Model Representations 4-1
General-Linear Polynomial Model 4-1
ARX Model 4-3
ARMAX Model 4-4
Output-Error Model 4-5
Box-Jenkins Model 4-6
AR Model 4-7
Transfer Function Model 4-8
Zero-Pole-Gain Model 4-9
State-Space Model 4-10
User-Defined Model 4-11
Polynomial Models versus State-Space Models 4-12
Determining Parameters for the Prediction Error Method 4-12
Akaike's Information Criterion 4-14
Final Prediction Error Criterion 4-14
Minimum Description Length Criterion 4-14

Chapter 5
Partially Known Model Estimation Methods

Comparing Unknown and Partially Known Model Estimation Methods 5-1
Defining and Estimating Partially Known Models 5-2
Setting Parameter Constraints with a Range 5-4
Setting Parameter Constraints with an Initial Guess 5-5
Partially Known Model Estimation Case Study 5-6

Chapter 6
Model Estimation Methods in Closed-Loop Systems

Feedback in a Closed-Loop System 6-1
Understanding Closed-Loop Model Estimation Methods 6-3
Direct Identification 6-5
Indirect Identification 6-6
Joint Input-Output Identification 6-8
Using System Identification VIs for Model Estimation 6-9

Chapter 7
Recursive Model Estimation Methods

Defining Recursive Model Estimation 7-1
Adaptive Algorithms 7-3
Least Mean Squares 7-4
Normalized Least Mean Squares 7-5
Recursive Least Squares 7-6
Kalman Filter 7-7

Chapter 8
Analyzing, Validating, and Converting Models

Analyzing Models 8-1
Bode Plot 8-2
Nyquist Plot 8-3
Pole-Zero Plot 8-4
Validating Models 8-5
Validation Methods 8-5
Model Simulation 8-6
Model Prediction 8-6
Residual Analysis 8-6
Converting Models 8-8

Chapter 9
System Identification Case Study

Data Preprocessing 9-1
Examining the Time Response Data 9-2
Examining the Frequency Response Data 9-3
Applying a Filter to the Raw Data 9-5
Downsampling the Raw Data 9-6
Estimating the Model 9-7
Akaike's Information Criterion 9-8
Verifying the Results 9-9
Minimum Description Length Criterion 9-10
User-Defined Criterion 9-12
ARX Model Validation 9-13
Simulation and Prediction 9-14
Residual Analysis 9-15
Estimating a State-Space Model 9-18
Finding the Singular Values 9-18
Validating the Estimated State-Space Model 9-19
Additional Examples 9-20

Appendix A
References

Appendix B
Technical Support and Professional Services

About This Manual

The LabVIEW System Identification Toolkit provides a library of VIs and an assistant for developing models of a system based on a large set of raw data. Both tools enable you to complete the entire system identification process from analyzing the raw data to validating the identified model.

This manual discusses the main steps in the system identification process and how to use the System Identification VIs to create applications that can accomplish the various tasks in the process. Refer to the *LabVIEW Help*, available in LabVIEW by selecting **Help»Search the LabVIEW Help**, for more information about the steps in the assistant and a tutorial about how to use the assistant.

The System Identification Assistant uses the same system identification concepts described in this manual. However, this manual does not include information about how to use the assistant. Refer to the *NI Express Workbench Help*, available in the NI Express Workbench environment by selecting **Help»Express Workbench Help**, for more information about the steps in the assistant and a tutorial about how to use the assistant.

Conventions

The following conventions are used in this manual:

» The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

This icon denotes a note, which alerts you to important information.

bold Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *LabVIEW Help*
- *LabVIEW Fundamentals*
- *Getting Started with LabVIEW*
- *NI Express Workbench Help*
- *LabVIEW System Identification Toolkit Algorithm References*
- *LabVIEW Control Design Toolkit User Manual*
- *Signal Processing Toolset User Manual*
- *Time Series Analysis Tools User Manual*

Refer to Appendix A, *References*, for a list of textbooks and technical papers that National Instruments used to develop the System Identification Toolkit.

1

Introduction to System Identification

System identification involves building mathematical models of a dynamic system based on a set of measured stimulus and response data samples. You can use system identification in a wide range of applications, including mechanical engineering, biology, physiology, meteorology, economics, and model-based control design. For example, engineers use a system model of the relationship between the fuel flow and the shaft speed of a turbojet engine to optimize the efficiency and operational stability of the engine. Biologists and physiologists use system identification techniques in areas such as eye pupil response and heart rate control. Meteorologists and economists build mathematical models based on historical data for use in forecasting.

This manual focuses on how to use system identification in the model-based control design process, which involves identifying a model of a plant, analyzing and synthesizing a controller for the plant, simulating the plant and controller, and deploying the controller. A plant is the real-world, physical system that you want to control.

System identification is the initial step—identifying a model of a plant—in the model-based control design process. System identification is an iterative process. You first acquire raw data from a real-world system, then format and process the data as necessary, and finally select a mathematical algorithm that you can use to identify a mathematical model of the system. You then can use the resulting mathematical model to analyze the dynamic characteristics and simulate the time response of the system. You also can use the mathematical model to design a model-based controller.

The LabVIEW System Identification Toolkit assists you in identifying large, multivariable models of high-order systems from large amounts of data. The System Identification Toolkit provides two tools, an assistant and a library of VIs, for identifying these linear systems. Both tools enable you to complete the entire system identification process, from analyzing raw data to validating the identified model.

This chapter provides an overview of the model-based control design process and the steps in the process where you can use National Instruments software and hardware. This chapter also describes various representations of physical models that you can use to identify the plant model of a system. Finally, this chapter provides information about the two tools in the System Identification Toolkit that enable you to identify system models.

Model-Based Control Design Process

The model-based control design process involves modeling a plant, analyzing and synthesizing a controller for the plant, simulating the plant and controller, and deploying the controller. While the System Identification Toolkit provides solutions for analyzing raw data and creating plant models, National Instruments also provides solutions for the other three components in the process, as shown in Figure 1-1.

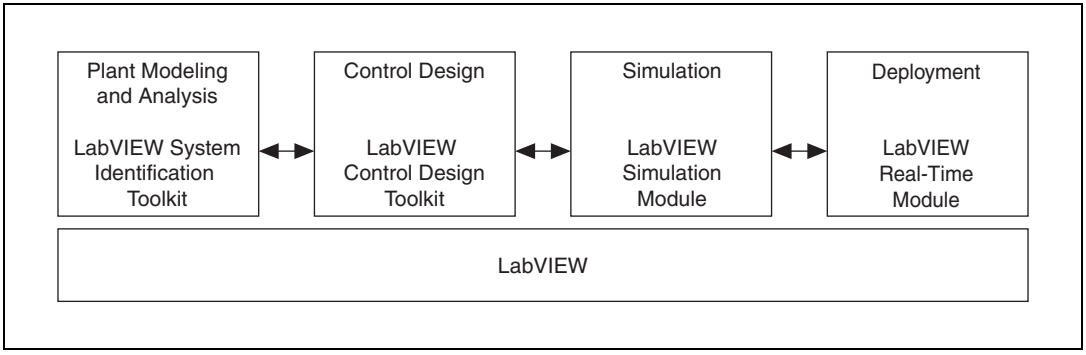

Figure 1-1. National Instruments Tools for the Model-Based Control Design Process

Analyzing Data and Creating a Dynamic System Model

In the initial phase of the design process, you must obtain a mathematical model of the plant you want to control. One way to obtain a model is by using a numerical process known as system identification. This process involves acquiring data from a plant and then numerically analyzing stimulus and response data to estimate the parameters of the plant.

National Instruments provides data acquisition (DAQ) and modular instrumentation software and hardware that you can use to stimulate and measure the response of the plant. You then can use the System Identification Toolkit to estimate and create accurate mathematical models of the plant.

System identification is a process that includes acquiring, preprocessing, and identifying mathematical models based on raw data from a real-world system. You then validate that the resulting model fits the observed system behavior. If the results are unsatisfactory, you revise the parameters and iterate through the process. Figure 1-2 shows a typical system identification flowchart.

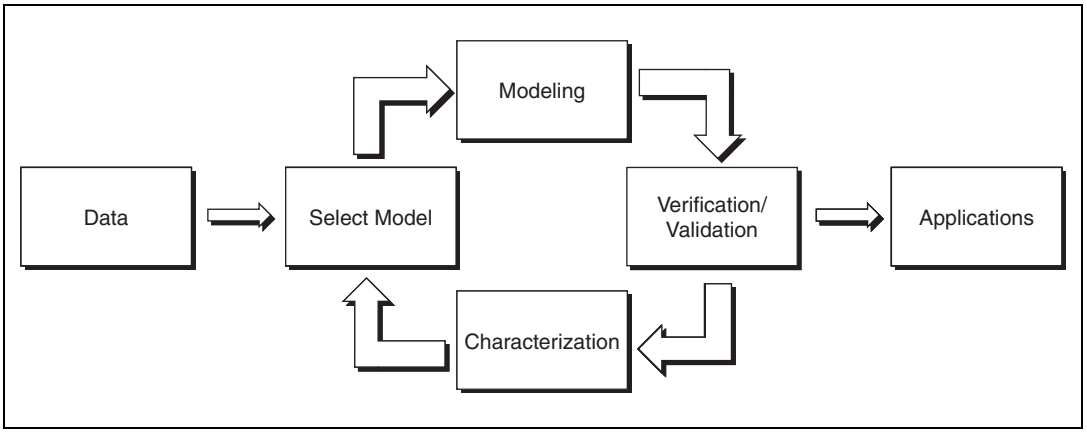

Figure 1-2. System Identification Application Flowchart

A real-world system seldom has one model that perfectly describes all the observed behaviors of the system. Because system identification involves many variables—such as sampling frequency, type of mathematical model, model order, and so on—you usually have a number of models you can use. Each model describes the behavior of the system to some extent or in a particular mode of operation.

Furthermore, multiple applicable algorithms might be available for the same model. The algorithms you select depend on the model structure, stochastic assumptions, and numerical properties of the algorithm. The System Identification Toolkit includes different adaptive techniques for recursive system identification and different algorithms for model estimation. Refer to Chapter 3, *Nonparametric Model Estimation Methods*, Chapter 4, *Parametric Model Estimation Methods*, Chapter 5, *Partially Known Model Estimation Methods*, Chapter 6, *Model Estimation Methods in Closed-Loop Systems*, and Chapter 7, *Recursive Model Estimation Methods*, for more information about the various estimation methods that the System Identification Toolkit supports.

Designing a Controller

In the second phase of the design process, you synthesize and analyze a controller. The LabVIEW Control Design Toolkit provides a set of VIs for classical and modern linear control analysis and design techniques. With these VIs you can create and analyze linear time-invariant (LTI) system models and design automatic control systems.

You can use the Control Design Toolkit to analyze the plant model you identified with the System Identification Toolkit. The Control Design VIs help you determine an appropriate controller structure. You then can synthesize a controller to achieve the desired performance criteria of the system based on the dynamic behavior of the plant and/or control system. Finally, you can analyze the overall system by combining the controller with the identified plant model.

Simulating the Dynamic System

In the third phase of the design process, you simulate the dynamic system. The LabVIEW Simulation Module allows you to simulate dynamic systems in LabVIEW. You can investigate the time response of the dynamic system to complex, time-varying inputs before deploying a controller. For this process, you can use a simple LTI model, a higher order model, or a nonlinear model of the plant.

Deploying the Controller

The last stage of the design process is to deploy the controller to a real-time target. LabVIEW and the LabVIEW Real-Time Module provide a common platform that you can use to implement or prototype the embedded control system. You also can use the Simulation Module and the Real-Time Module as the platform for implementing the control system.

National Instruments also provides products for I/O and signal conditioning that you can use to gather and process data. Using these tools, which are built on the LabVIEW platform, you can experiment with different approaches at each stage in the design process and quickly identify the optimal design solution for an embedded control system.

Refer to the National Instruments Web site at ni.com for more information about these National Instruments products.

Model Types and Model Representations

You can represent a dynamic system using several types of dynamic system models. The following sections provide information about the different types and representations of dynamic system models that you can use with the System Identification Toolkit.

Model Types

You base the type of a dynamic system model on the properties of the dynamic system that the model represents. The following sections provide information about the different types of models you can create with the System Identification Toolkit.

Linear versus Nonlinear Models

Dynamic system models are either linear or nonlinear. A linear model obeys the principles of superposition and homogeneity, as shown by Equations 1-1 and 1-2, respectively.

$$y_1 = f(u_1)$$

$$y_2 = f(u_2)$$

$$f(u_1 + u_2) = f(u_1) + f(u_2) = y_1 + y_2 \quad (1\text{-}1)$$

$$f(a_1 u_1) = a_1 f(u_1) = a_1 y_1 \quad (1\text{-}2)$$

where u_1 and u_2 are the system inputs and y_1 and y_2 are the system outputs.

Conversely, nonlinear models do not obey the principles of superposition or homogeneity. Nonlinear effects in real-world systems include saturation, dead-zone, friction, backlash, and quantization effects; relays; switches; and rate limiters. Many real-world systems are nonlinear, but you can simulate most real-world systems with linear models to simplify a design or analysis procedure.

Time-Variant versus Time-Invariant Models

Dynamic system models are either time-variant or time-invariant. The parameters of a time-variant model change with time. For example, you can use a time-variant model to describe an automobile. As fuel burns, the mass of the vehicle changes with time.

Conversely, the parameters of a time-invariant model do not change with time. For an example of a time-invariant model, consider a simple robot. Generally, the dynamic characteristics of robots do not change over short periods of time.

Continuous versus Discrete Models

Dynamic system models are either continuous or discrete. Both continuous and discrete system models can be linear or nonlinear and time-invariant or time-variant. Continuous models describe how the behavior of a system varies continuously with time, which means you can obtain the properties of a system at any certain moment from the continuous model. Discrete models describe the behavior of a system at separate time instants, which means you cannot obtain the behavior of the system between every two sampling points.

Continuous system models are analog. You derive continuous models of a physical system from differential equations of the system. The coefficients of continuous models have clear physical meanings. For example, you can derive the continuous transfer function of an RC circuit if you know the details of the circuit. The coefficients of the continuous transfer function are the functions of *R* and *C* in the circuit. You use continuous models if you need to match the coefficients of a model to some physical components in the system.

Discrete system models are digital. You derive discrete models of a physical system from difference equations or by converting continuous models to discrete models. In computer-based applications, signals and operations are digital. Thus, you can use discrete models to implement a digital controller or to simulate the behavior of a physical system at discrete instants. You also can use discrete models in the accurate model-based design of a discrete controller for a plant.

Model Representations

You can use the System Identification Toolkit to represent dynamic system models in the following four categories—general-linear polynomial, transfer function, zero-pole-gain, and state-space. Refer to Chapter 4, *Parametric Model Estimation Methods*, for more information about these models.

Developing Models with the System Identification Toolkit

The System Identification Toolkit provides a library of VIs and an assistant. You can apply these VIs or the assistant to develop and validate the plant model of a system that uses a large set of raw data.

System Identification VIs

The System Identification Toolkit provides VIs that you can use to preprocess raw data from a dynamic system and develop a model that reflects the behavior of that system. The Data Preprocessing VIs enable you to analyze the response of a plant or dynamic system to a certain stimulus. After analyzing the data, you can use the Parametric Model Estimation, Nonparametric Model Estimation, Partially Known Model Estimation, or Recursive Model Estimation VIs to estimate a model for the plant or dynamic system. Finally, you can use the Model Validation or Model Analysis VIs to determine whether the model accurately describes the dynamics of the identified system.

The System Identification VIs enable you to customize a LabVIEW block diagram to achieve specific goals. You also can use other LabVIEW VIs and functions to enhance the functionality of the application. Creating a LabVIEW application using the System Identification VIs requires basic knowledge about programming in LabVIEW. Refer to the *LabVIEW Fundamentals* and *Getting Started with LabVIEW* manuals for more information about the LabVIEW programming environment.

System Identification Assistant

If you do not have prior knowledge about programming in LabVIEW, you can use the System Identification Assistant to develop a model that reflects the behavior of a certain dynamic system. You access the System Identification Assistant through the NI Express Workbench. The Express Workbench is a framework that can host multiple interactive National Instruments tools and assistants.

Using the System Identification Assistant, you can create a project that encompasses the whole system identification process. In a single project, you can load or acquire raw data into the System Identification Assistant, preprocess the data, estimate a model that describes the system, and then validate the accuracy of the model. The Express Workbench provides windows in which you can see the raw data, the response data, the

estimated model, the validation results, and the mathematical equations that describe the model.

After creating a project in the Express Workbench, you can convert the project to a LabVIEW block diagram and customize the block diagram in LabVIEW. This conversion enables you to enhance the capabilities of the application. Refer to the *NI Express Workbench Help*, available in the NI Express Workbench environment by selecting **Help»Express Workbench Help**, for more information about using the assistant to develop models.

System Identification Procedure

You first must acquire data before estimating a model for the plant in a system. Then you can preprocess raw data by using a number of data preprocessing techniques the System Identification Toolkit provides to get high-quality data.

When data is collected, you can use nonparametric model estimation methods to estimate the impulse response and the frequency response of a system. Nonparametric model estimation is often less accurate. However, you can use a nonparametric model estimation method to obtain useful information about a system before applying parametric model estimation, which provides more insight into a system. Also, if you have some information about the system, you can set constraints on parameters by using partially known model estimation methods the System Identification Toolkit provides. When you have a plant in a closed-loop system, you can use the VIs this toolkit provides to identify the plant. You also can use recursive model estimation methods with the System Identification Toolkit to update a system model while the system is running.

After you estimate a model, you can analyze the model to obtain some information, such as frequency response, about the model and investigate model estimation results. You also can validate an estimated model to determine how accurately the model describes the real-world plant of a system.

With the System Identification Toolkit, you can convert system models from one representation to another and convert between continuous and discrete models. You have the flexibility to obtain best-fit models in different applications.

The following chapters describe data acquisition and preprocessing techniques, model estimation methods, model analysis and validation techniques, and model conversion tools.

2

Acquiring and Preprocessing Data

The first step in identifying an unknown system is data acquisition. You can acquire data from NI data acquisition hardware and software or you can use data from a pre-stored file. For verification and validation reasons, you need to acquire two sets of input-output data samples or split the data into two sets. You use one set of samples to estimate the mathematical model of the system. You use the second set of samples to validate the resulting model. If the resulting model does not meet the predefined specifications, such as the mean square error (MSE), modify the settings and re-verify the resulting model with the data sets.

After acquiring the data, you need to preprocess the raw data samples. Preprocessing involves steps such as removing trends, filtering noise, and so on. The LabVIEW System Identification Toolkit provides Data Preprocessing VIs that enable you to analyze the raw data and determine whether the data accurately reflects the response of the system you want to identify.

This chapter briefly describes the data acquisition process and the assumptions the System Identification Toolkit makes. This chapter also describes how to preprocess raw data using the Data Preprocessing VIs. Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for more information about the Data Preprocessing VIs.

Acquiring Data from a System

One of the biggest advantages of using the System Identification Toolkit is the integration with LabVIEW, NI data acquisition hardware, and NI-DAQ. Refer to the *LabVIEW Help*, available in LabVIEW by selecting **Help»Search the LabVIEW Help**, for more information about setting up and configuring a data acquisition system, and how to use LabVIEW to acquire data samples.

Identifying a system involves a number of choices with regard to the system output signals you want to measure and the input signals you want to manipulate. The choices you make about how to manipulate system inputs, types of signal conditioning, signal ranges, and sampling behavior affect the validity of the model you obtain. You can use different modeling techniques on the same experimental data set, but if the data set does not reflect the behavior of interest then you need to acquire a more descriptive data set.

Because the system identification process is often an experimental process, it is often time consuming and possibly costly. Therefore, you must think about the design of process prior to experimenting with various identification techniques. The following sections describe the various data acquisition and system stimulation assumptions you must consider before identifying a system model. These sections also provide information about the trade-offs associated with each choice.

Accounting for Factors that Influence a System

The key to the system identification process is having some knowledge of the system for which you want to identify a model. This knowledge provides the basis for determining which signals are outputs, which in turn determines sensor placement, and which signals are inputs that you can use to excite the system. Simple tests might be necessary to determine influences, coupling, time delays, and time constants to aid in the modeling effort.

Also you need to consider signals that are not directly capable of being manipulated but still affect the system. You need to include those signals as inputs to the system model. For example, consider the effect of wind gusts on the pitch dynamics of an airplane. The airplane responds in pitch to the elevator angle as a direct input. A wind gust affects the pitch of an airplane, which in turn influences the dynamics of the airplane, but the wind gust is not directly adjustable. To create an accurate model of the airplane, you might want to include wind gusts as an input variable.

Choosing a Stimulus Signal

The choice of stimulus signals has an important role in the observed system behavior and the accuracy of the estimated model. These signals determine the operating points of the system. While the system under test often limits the choice of signals, you want an input signal to exhibit certain characteristics to produce a response that provides the information needed

for developing an accurate model. The following sections summarize these characteristics.

- To obtain meaningful dynamic behavior, you must test the system under conditions similar to the actual operating conditions. When you complete experiments in these conditions, you identify the system in the same conditions under which you will implement the resulting model. This criterion is extremely important for nonlinear systems.
- You want the inputs to the system under test to excite the system. Exciting the system is dependent on the spectrum of the input signal. Specifically, you must excite the system with an input frequency similar to the frequency at which such inputs change during normal operations.
- You want the amplitude of the step input to cover a wide range of variations. Therefore, in the data you use for model estimation, you need to cover the normal operation range of system inputs, especially when you use the calculated model for model-based control. To cover the normal operation range, you can combine the positive and negative step changes of different magnitudes in the system inputs.
- You want the input signal to deliver as much input power to the system as possible. However, in the real-world, you must ensure that this input power stays within the limits of the physical system. The crest factor C_f, defined by the following equation, describes this property.

$$C_f^2 \equiv \frac{\max_t u^2(t)}{\lim_{N\to\infty}\frac{1}{N}\sum_{t=1}^{N} u^2(t)}$$

  The smaller the crest factor the better the signal excitation resulting in larger total energy delivery and enhanced signal-to-noise ratio. The theoretical lower bound for crest factor is 1.

Common Stimulus Signals

The system response data is dependent on the physics of the system you want to study. Some systems tend to respond faster than others, and never reach steady state. Other systems have large time constants and delays. For these reasons, defining a stimulus signal that provides enough excitation to the system is important, so that the response captures the important features of the system dynamics. The following sections describe common stimulus signals you can use in different process applications.

Filtered Gaussian White Noise

Filtered Gaussian white noise is a simple signal that can generate virtually any signal spectra in conjunction with the proper linear filtering. The theoretical crest factor C_f for a Gaussian is infinite, but clipping the Gaussian amplitude to the input signal limits results in a corresponding reduction in crest factor while minimally affecting the generated spectrum.

Figure 2-1 shows an example of a Filtered Gaussian white noise.

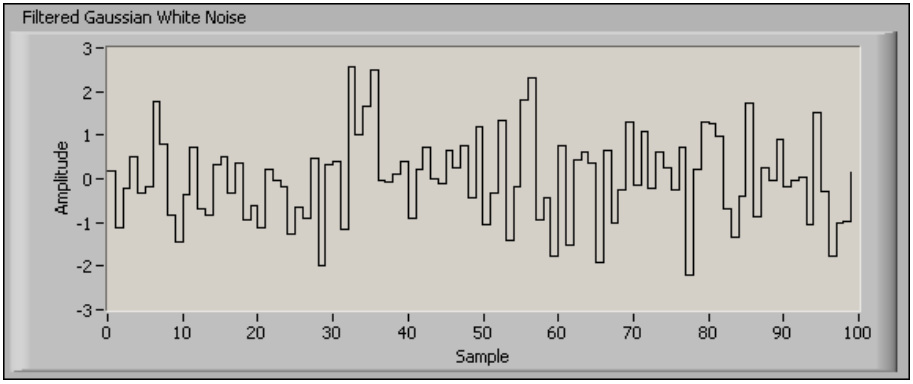

Figure 2-1. Filtered Gaussian White Noise

Random Binary Signal

A random binary signal is a random process that can assume one of two possible values at any time. A simple method of generating a random binary signal is to take Gaussian white noise, filter it for the desired spectra and then convert it to a binary signal by taking the sign of the filtered signal. The desired spectra is a function of the system time constraints. The appropriate scaling must provide a meaningful response to the system, well above the noise level.

You can scale the signal to any desired amplitude. The resulting signal has a minimum crest factor C_f of 1. Some differences in the resulting spectra are expected so you must perform off-line analysis of the signal.

Binary signals are useful for identifying linear systems. However, the dual-level signal does not allow for validation against nonlinearities. If a system is nonlinear, you can use an input interval corresponding to the desired operating point. You might need to work with more than two input levels in these cases. You can combine multiple binary signals of different levels to form the stimulus signal.

Figure 2-2 shows an example of a random binary signal.

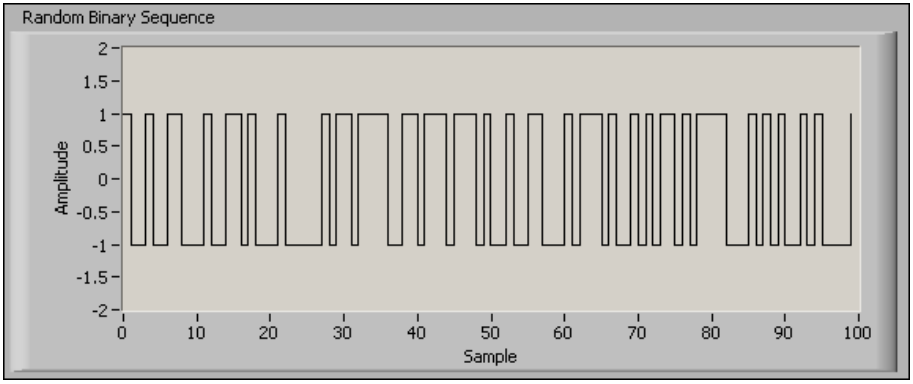

Figure 2-2. Random Binary Sequence

Pseudo-Random Binary Sequence

A Pseudo-Random Binary Sequence, also known as Maximal Length Sequence (MLS), is a periodic, deterministic signal with properties similar to white noise. You often generate a pseudo-random binary sequence using an n-bit shift register with feedback through an exclusive-OR function. While appearing random, the sequence actually repeats every $2n - 1$ values.

When using a whole period, the pseudo-random binary sequence has special mathematical advantages that make it attractive as a stimulus signal. In particular, you can attribute variations in response signals between two periods of the stimulus to noise due to the periodic nature of the signal. Also, like the white random binary noise, the pseudo-random binary sequence has a low crest factor C_f. You can use the SI Generate Pseudo-Random Binary Sequence VI to generate a Pseudo-Random Binary Sequence.

Figure 2-3 shows an example of a pseudo-random binary sequence.

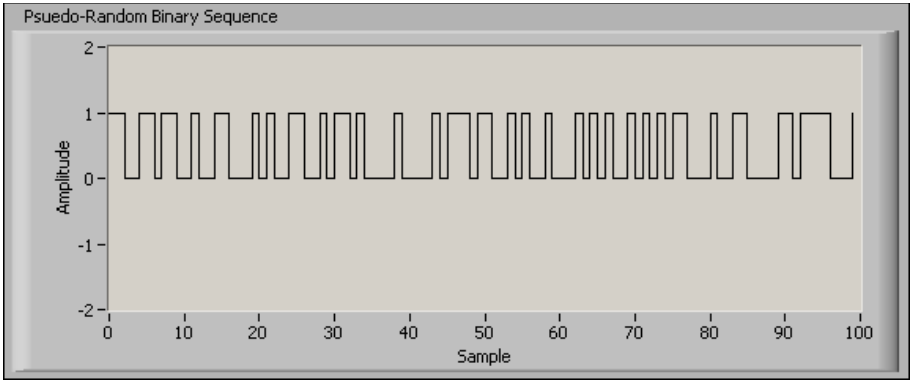

Figure 2-3. Pseudo-random Binary Sequence

Chirp Waveform

The chirp waveform, also known as a swept sine wave, is a sinusoid waveform with a frequency that varies continuously over a certain range of values $\omega_1 \le \omega \le \omega_2$ for a specific period of time $0 \le t \le T$. The resulting signal has a crest factor C_f of $\sqrt{2}$. You can modify the signal to excite specific signal spectra.

In comparison to other signals, like the white noise stimulus, a chirp waveform is easier to generate and control. Figure 2-4 shows an example of a chirp waveform.

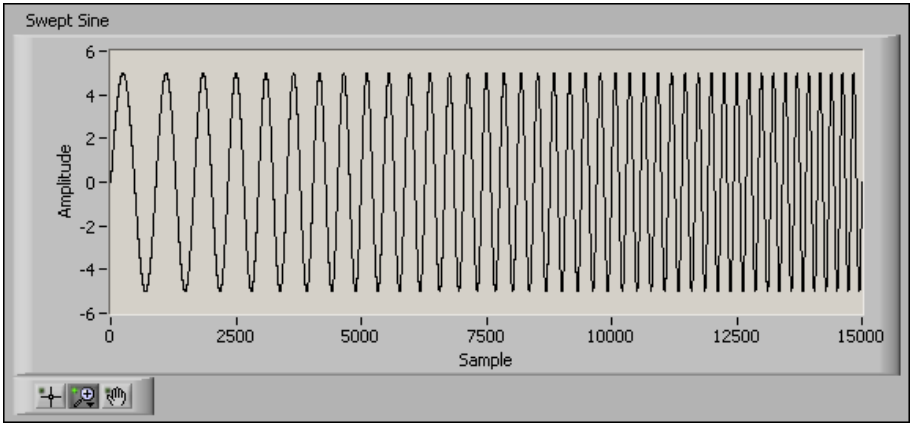

Figure 2-4. Chirp, or Swept Sine Wave

Selecting a Sampling Rate

The time constants of a system influence the selection of a sampling rate. Sampling at rates substantially greater than the system bandwidth leads to data redundancy, numerical issues, and modeling of high frequency artifacts likely due to noise. Sampling at rates slower than system dynamics leads to difficulties determining an accurate system model and problems introduced by aliasing. You can use an anti-aliasing filter to counter the effects of aliasing. Refer to the *Applying an Anti-Aliasing Filter* section for more information about anti-aliasing filters.

A common rule of thumb is to sample signals at 10 times the bandwidth of the system or the bandwidth of interest for the model. If uncertainty exists in the system bandwidth and a fast data acquisition environment is available, you can sample as fast as possible, then use a digital filter and decimation to reduce the sampling rate to the desired value. Decimation is a form of downsampling the data set. Refer to the *Filtering and Downsampling* section for more information about filtering and downsampling a data set.

Applying an Anti-Aliasing Filter

According to the Nyquist sampling theorem, the sampling rate must be greater than twice the maximum frequency component of the signal of interest. In other words, the maximum frequency of the input signal must be greater than half the sampling rate.

This criterion, in practice, is often difficult to ensure. Even if you are sure that the measured signal has an upper limit on its frequency, external factors such as signals from the powerline interference or radio stations, can contain frequencies higher than the Nyquist frequency. These frequencies might then alias into the frequency range of interest and give you inaccurate results.

To ensure that you limit the frequency content of the input signal, add a lowpass filter before the sampler and the analog to discrete converter. A lowpass filter passes low frequencies and attenuates high frequencies. This filter is an anti-aliasing filter because by attenuating the frequencies greater than the Nyquist frequency, the filter prevents the sampling of aliased components. When you use a filter before the sampler and analog to discrete converter, the anti-aliasing filter is an analog filter. Using an analog filter satisfies the Nyquist sampling theorem.

Similarly, you can use a digital filter to remove frequency content above the system bandwidth and then decimate or downsample the data to the desired sampling rate.

Preprocessing Data from a System

A number of preprocessing techniques ensure that the incoming data samples are free from external noise, scaling problems, outliers, and other corruptions. These preprocessing techniques include the following methods:

- Visually inspecting data
- Removing offsets and trends
- Removing outliers
- Filtering and downsampling

Validating the quality of the data at each step in the preprocessing procedure is important in ensuring that you accurately identify a model in later steps of the system identification process.

The following sections describe these preprocessing techniques and how you can use the System Identification Toolkit to apply these techniques.

Visually Inspecting the Data

Visual inspection of the data is the best way to detect major signal corruptions or errors—such as outliers, clipped saturation, or quantization effects—that occur during acquisition or preprocessing. You also can plot the data waveform and the spectral density function of the data to discover periodic disturbances.

Traditionally, you examine data samples either in the time domain or the frequency domain. An effective approach is to display the data in the joint time-frequency domain, which provides a better understanding about the measured signals. Refer to the *Signal Processing Toolset User Manual*, available at `ni.com/manuals`, for more information about joint time-frequency domain techniques for data processing.

Removing Offsets and Trends

The SI Remove Trend VI enables you to remove offsets and trends from the raw data set. You can specify which you want to remove using the input **trend type**. The following sections describe the difference between removing offsets and removing trends.

Removing Offsets

The estimated system model is a linearized version of the true system around the operating point. You must subtract the operating points from the raw data samples because linearization is done with respect to the signal values relative to the operating point, which is the offset level of the signal.

Figure 2-5 shows an example of removing the offset level of a signal. The goal of the water tank is to keep the water level at six meters. The **Water level record** graph shows that the water level changes in the vicinity of the operating point of six meters. If you use the water level record for system identification, you must remove the six meter operating point value.

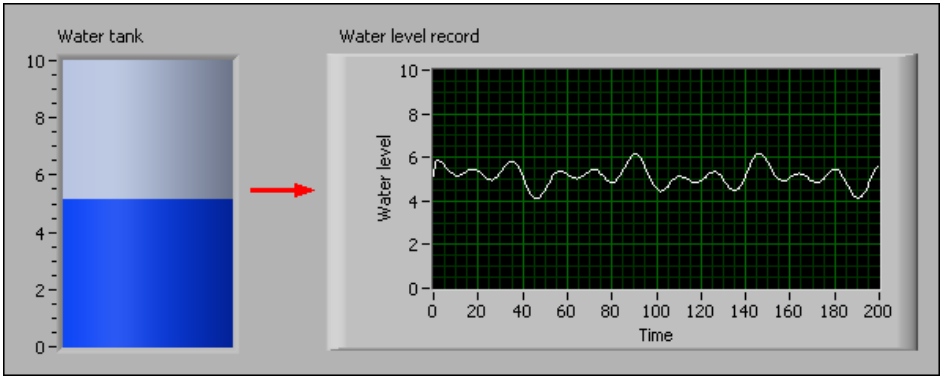

Figure 2-5. Operating Point of Water Tank

The SI Remove Trend VI enables you to remove the offset from the raw data set. You must set the **trend type** to **mean** to use this preprocessing technique.

Removing Trends

External influences might add some low frequency or periodic components, which are not relevant to the specific modeling problem, to the data. Examples of external influences include variations due to the 24-hour day cycle in power plants, seasonal influences in biological and economical systems, thermal expansion in rolling mills, 50 Hz and 60 Hz powerline interferences, and so on. The amplitude of these trends can be large and can corrupt the results of signal analysis and parametric identification algorithms.

The SI Remove Trend VI provides a way for you to remove these external influences, or trends, from the raw data set. You must set the **trend type** to **linear** to use this preprocessing technique.

Outliers

Various unexpected events, such as an abnormal pulse, a temporary sensor failure, or transmitter failure, can corrupt the raw data samples. These disturbances, or outliers, can severely distort the resulting model estimation. However, you often can recognize outliers by visually inspecting the data, as shown in Figure 2-6.

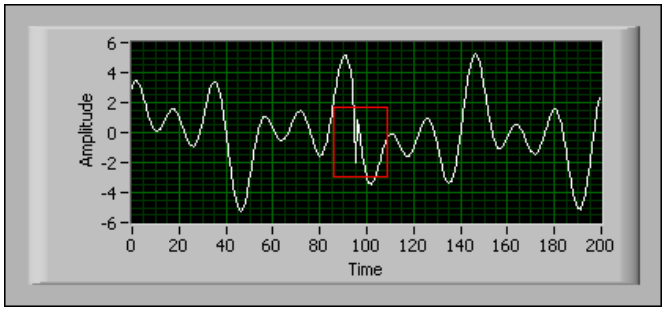

Figure 2-6. Data with Outliers

Visually inspecting the graph tells you that the data acquired between 85–100 seconds is abnormal. When preprocessing data, you want to remove all outliers in the data set. You must remove the outliers manually.

Filtering and Downsampling

You might be interested in only a specific frequency range of the frequency response for a model. You can filter and enhance the data in the frequency range to improve the fit in the regions of interest. If the sampling frequency is much higher than the bandwidth of the system, the sampling frequency might substantially increase the computation burden for complicated identification algorithms. You can decrease the sampling frequency by taking every n^{th} sample to construct a new downsampled data set. Applying an anti-alias filter on the data before downsampling prevents corruption of the downsampled data set.

You can use the SI Lowpass Filter VI or the SI Bandpass Filter VI to apply a lowpass or bandpass filter, respectively, to the data from the system. You then can use the SI Downsampling VI to reduce the number of samples in the data set.

After preprocessing the data you acquired from a dynamic system, the result is a data set that you can use to estimate a model that reflects the system dynamics. Refer to Chapter 3, *Nonparametric Model Estimation Methods*, for more information about the nonparametric model estimation

methods that use the impulse response and frequency response. Refer to Chapter 4, *Parametric Model Estimation Methods*, for more information about the parametric model structures and the parametric model estimation methods. Refer to Chapter 7, *Recursive Model Estimation Methods*, for more information about recursive model estimation methods.

Data Scaling

In multiple-input multiple-output (MIMO) systems, to have inputs and outputs of different amplitude ranges is common. Such a diversity in amplitudes can make the model estimation calculation ill-conditioned, which deteriorates the precision of the dynamic response. For example, consider the values A and B in Figure 2-7. Valves A and B operate between 0–100% and 50–60% opening range, respectively.

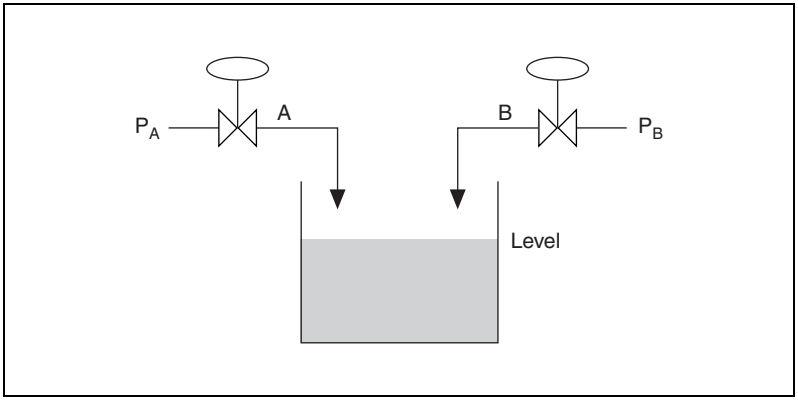

Figure 2-7. Tanks

The pressure in their respective stream lines are P_A and P_B. Assume that P_B can be much larger than P_A, you might need to normalize the range of operation of valve B for numerical robustness. You can use the following relationship to normalize the range of operation.

$$\frac{\Delta Level}{\Delta \%A} \approx \frac{\Delta Level}{(\Delta \%B - 50)10}$$

The SI Normalize VI ensures that all stimulus and response signals have a zero mean and unit variance over the sample data range used for model estimation. This process standardizes the range of the equation for all signals considered for model estimation. This data preprocessing step considers all inputs and outputs equally important from the numerical calculation viewpoint.

3

Nonparametric Model Estimation Methods

After acquiring and preprocessing the data from a linear time-invariant system, the next step in the system identification process is to estimate the model. The two most common techniques to estimate models that represent linear time-invariant systems are nonparametric estimation and parametric estimation. This chapter describes the nonparametric estimation methods.

You can describe linear time-invariant models with transfer functions or by using the impulse responses or frequency response of the system. The impulse response and frequency response are two ways of estimating a nonparametric model. The impulse response reveals the time-domain properties of the system, such as time delay and damping, whereas the frequency response reveals the frequency-domain properties, such as the natural frequency of a dynamic system.

Nonparametric model estimation is simple and more efficient, but often less accurate, than parametric estimation. However, you can use a nonparametric model estimation method to obtain useful information about a system before applying parametric model estimation. For example, you can use nonparametric model estimation to determine whether the system requires preconditioning, what the time delay of the system is, what model order to select, and so on. You also can use nonparametric model estimation to verify parametric models. For example, you can compare the Bode plot of a parametric model with the frequency response of the nonparametric model. Refer to Chapter 4, *Parametric Model Estimation Methods*, for more information about parametric model estimation methods.

The LabVIEW System Identification Toolkit uses least squares method and correlation analysis method to estimate the impulse response and spectral analysis method to estimate the frequency response. The following sections describe the impulse response and frequency response methods.

Impulse Response

An impulse input, as shown in Figure 3-1, to a dynamic system is defined differently depending on whether the system is discrete or continuous. For a continuous dynamic system, an impulse input is a unit-area signal with an infinite amplitude and infinitely small duration occurring at a specified time. At all other times, the input signal value is zero. For a discrete system, an impulse is a physical pulse that has unit amplitude at the first sample period and zero amplitude for all other times.

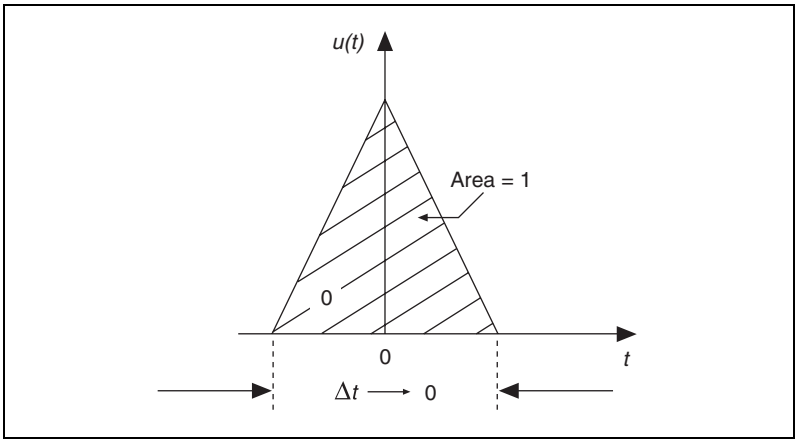

Figure 3-1. Impulse Response

Because the impulse signal excites all frequencies and the duration of this signal is infinitely small, you can see the natural response of the system.

Figure 3-2 shows that the impulse response of a linear time-invariant system is equal to the output $y(k)$ of the system when you apply an impulse signal to the input $u(k)$ of the system. The impulse response provides the complete characteristic information of a system.

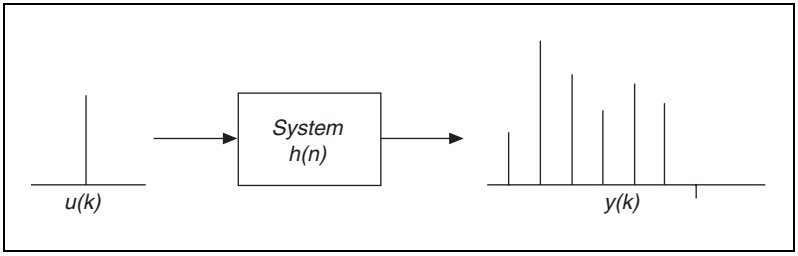

Figure 3-2. Impulse Response Definition

If you know the impulse response $h(n)$ and the input signal $u(k)$ of a system, then you can compute the output $y(k)$ of the system by using the following equation.

$$y(k) = \sum_{k=-\infty}^{\infty} u(k-n)h(n) + e(k) \quad (3\text{-}1)$$

where $e(k)$ is the disturbance of the system.

According to impulse response theory, when you apply a Dirac delta function to the input of a system, the output of the system is the impulse response. You can think of the Dirac delta function $\delta(x)$ as a function that has the value of infinity for $x = 0$, the value zero elsewhere, and a total integral of one. However, generating an ideal Dirac delta function is unrealistic.

If you apply an approximate impulse with a small duration to the input of a system, the output of the system is the approximation of the impulse response of the system. The smaller the duration of the impulse, the closer the output of the system is to the true impulse response. However, an impulse carries little energy and might not excite the system, and noise might corrupt the output of the system. An impulse with a large amplitude and duration can improve the signal-to-noise ratio of the output signal. However, the large amplitude impulse can damage the hardware of the system, and a long-duration impulse leads to inaccuracy. For these reasons, the System Identification Toolkit uses the least squares method and the correlation analysis method to estimate the impulse response.

Least Squares

If both the input signal $u(k)$ and output signal $y(k)$ of a system are available, you can obtain the value of $h(k)$, as shown in Equation 3-1. This method does not require a Dirac delta function as the input signal of the system. Instead, you can use common stimulus signal and the corresponding response signal from the system to compute the impulse response mathematically. You can obtain the impulse response for both positive and negative lags.

The Least Squares instances of the SI Estimate Impulse Response VI implements the least squares method to solve the equation. Refer to the *LabVIEW System Identification Toolkit Algorithm References* manual (`SIreference.pdf`), available in the `labview\manuals` directory, for more information about the least squares method.

Correlation Analysis

The correlation analysis method uses the cross correlation between the input and output signals as an estimation of the impulse response, as shown by the following equation:

$$y(k) = \sum_{k=0}^{\infty} u(k-n)h(n) + e(k)$$

The input signal must be zero-mean white noise with a spectral density that is equally distributed across the whole frequency range. The SI Estimate Impulse Response VI can prewhiten input signals that are not white noise.

Assuming the input $u(k)$ of the system is a stationary, stochastic process and statistically independent of the disturbance $e(k)$, the following equation is true.

$$R_{uy}(\tau) = \sum_{k=0}^{\infty} R_{uu}(k-\tau)h(k)$$

R_{uy} represents the cross correlation function between the stimulus signal $u(k)$ and the response signal $y(k)$, as defined by the following equation.

$$R_{uy}(\tau) = \frac{1}{N} \sum_{k=min(\tau,0)}^{N-max(\tau,0)-1} y(k+\tau)u(k)$$

R_{uu} represents the autocorrelation of the stimulus signal $u(k)$, as defined by the following equation.

$$R_{uu}(\tau) = \frac{1}{N} \sum_{k=0}^{N-\tau-1} u(k+\tau)u(k)$$

N is the number of data points. If the stimulus signal is a zero-mean white noise signal, the autocorrelation function reduces to the following equation.

$$R_{uu}(\tau) = \sigma_u^2 \delta(\tau)$$

where σ_u is the standard deviation of the stimulus white noise and $\delta(\tau)$ is the Dirac function. Substituting $R_{uu}(\tau)$ into the cross correlation function

between the stimulus signal $u(k)$ and the response signal $y(k)$ yields the following equation.

$$R_{uy}(\tau) = \sigma_u^2 \sum_{k=0}^{\infty} \delta(k-\tau)h(k) = \sigma_u^2 h(\tau)$$

You can rearrange the terms of this equation to obtain the following equation defining the impulse response $h(k)$.

$$h(k) = \frac{R_{uy}(k)}{\sigma_u^2}$$

Prewhitening

The correlation analysis method that estimates the impulse response is useful only when the input signal $u(k)$ is a zero-mean white noise signal. However, the input signal is not white noise in most real-world applications. The input $u(k)$ and output $y(k)$ signals therefore must be preconditioned before you apply the correlation analysis method.

Prewhitening is a preconditioning technique for the correlation analysis method. Prewhitening involves applying a filter to the input signal $u(k)$ and the output signal $y(k)$ to obtain a prewhitened input signal $u'(k)$ and a prewhitened output signal $y'(k)$. If the filter is well designed such that $u'(k)$ is white noise, you can perform a correlation analysis on $u'(k)$ and $y'(k)$ to estimate the impulse response. The impulse response that you estimate with $u'(k)$ and $y'(k)$ is equivalent to the impulse response that you estimate with $u(k)$ and $y(k)$ because the following equation remains true.

$$y'(k) = \sum_{n=0}^{\infty} u'(k-n)h(n) + e(k)$$

You now must design the prewhitening filter so that $u'(k)$ is white noise. The SI Estimate Impulse Response VI uses an AR model for this purpose. Refer to Chapter 4, *Parametric Model Estimation Methods*, for more information about AR model estimation.

Accuracy of the Impulse Response

The accuracy of the impulse response estimation using the correlation analysis method depends on the performance of the prewhitening filter, specifically whether the filter produces a white noise result $u'(k)$ for $u(k)$. The performance of the filter depends on the signal and the AR order of the filter. The rule of thumb for selecting the AR order is trial-and-error. If $u'(k)$ is not white enough, the result from the correlation method is not reliable. You can increase the AR order to improve the accuracy of the impulse response.

The SI Estimate Impulse Response VI provides the outputs **whiteness test** and **rejected?** to indicate whether you have properly set the AR order and consequently whether the impulse response estimation is reliable. The following example shows how the whiteness property of the input signal affects the correlation analysis method and how to use the outputs **whiteness test** and **rejected?** to justify the impulse response estimation.

Figure 3-3 shows the front panel of a VI that simulates a system defined by the following equation.

$$y(k) = 0.2u(k) + 0.8u(k-1) + 0.3u(k-2)$$

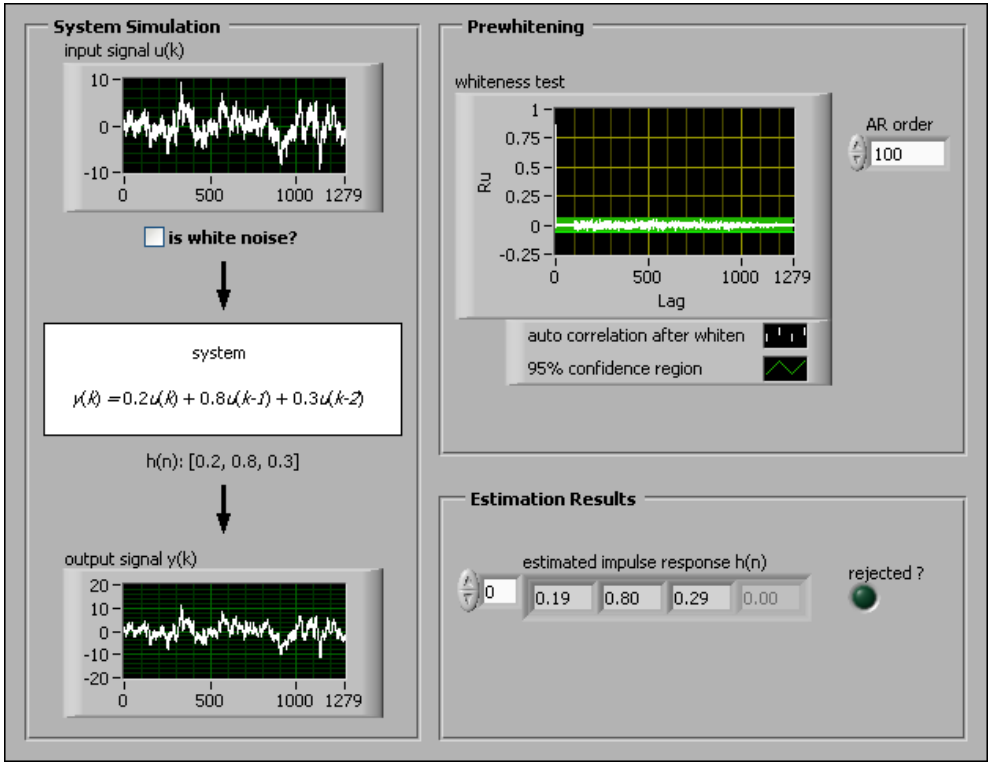

Figure 3-3. Front Panel of Prewhitening Example VI

Figure 3-4 shows the block diagram of this VI. This example VI demonstrates the accuracy of the impulse response estimation in the following circumstances:

- Zero-mean, pseudo-white noise input signal without prewhitening
- Zero-mean, pseudo-white noise input signal with prewhitening
- Non-zero-mean, white noise input signal without prewhitening
- Non-zero-mean white noise input signal with prewhitening

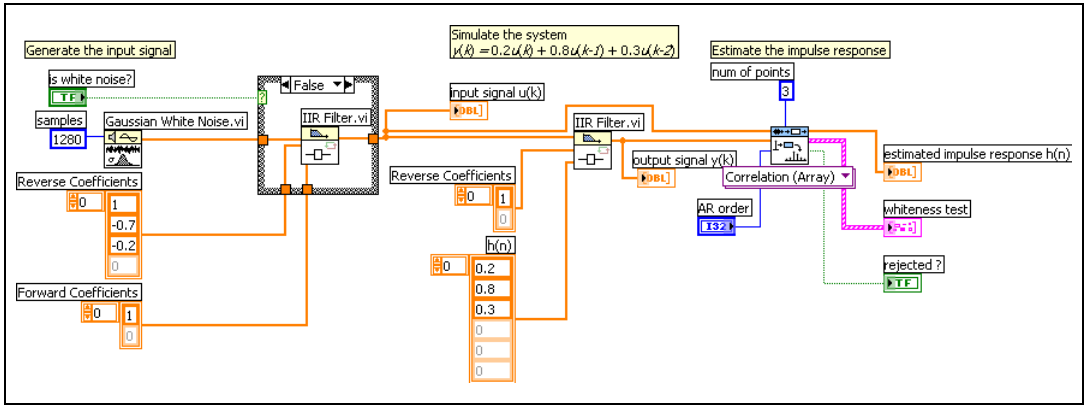

Figure 3-4. Block Diagram of Prewhitening Example VI

In this example VI, the **is white noise?** checkbox determines whether the SI Estimate Impulse Response VI generates zero-mean white noise as an input to the system. When you place a checkmark in the **is white noise?** checkbox and run the VI, the generated input signal is zero-mean white noise, and the estimated impulse response closely approximates the true impulse response. When you do not place a checkmark in the **is white noise?** checkbox, the generated input signal is not zero-mean white noise. As a result, the estimated impulse response is different from the true impulse response. These results indicate that the correlation analysis method is accurate and reliable when the input signal is zero-mean white noise.

The **AR order** box determines the level of prewhitening. When **AR order** equals 0, the SI Estimate Impulse Response VI does not apply prewhitening to the system. When **AR order** is small and you do not place a checkmark in the **is white noise?** checkbox, the variance of the impulse response is large because the input signal is not always white noise. The greater the value of **AR order**, the better the VI whitens the signal, but the more computation time and memory the VI requires.

The **whiteness test** indicator of this VI shows whether the input is zero-mean white noise. This indicator displays the autocorrelation of the stimulus signal after whitening. If most of the autocorrelation is within the confidence region, the input signal is well prewhitened, and the estimation of the impulse response is reliable. If the autocorrelation is outside of the confidence region, the estimation is unreliable. When the estimation is unreliable, **rejected?** is TRUE and indicates a 5% risk of rejecting an impulse response estimation that might be reliable.

If you apply proper prewhitening, the correlation analysis method is accurate and reliable for any input signal. To obtain the best prewhitening settings, start with a small **AR order** value like 2 and observe the **whiteness test** and **rejected?** outputs of the SI Estimate Impulse Response VI. If necessary, increase the value of **AR order**. Generally, the smaller the bandwidth of the input signal, the larger the **AR order** you need. However, avoid setting the value of **AR order** greater than 500.

Selecting Impulse Response Length

Theoretically, the length of the impulse response might be infinite. For some systems, the impulse response quickly reaches zero, and the number of nonzero points is finite. For other systems, the impulse response never reaches zero. Realistically, you only can obtain the first *N* points of the impulse response due to limited signal length and limited memory size. Therefore, the SI Estimate Impulse Response VI has inputs to specify how many points of the impulse response to observe. With the least squares method, you must ensure the sum of **num of points (t<0)** and **num of points (t>=0)** is no larger than the signal length. With the correlation analysis method, you can set **num of points** to be as large as the signal length.

Applications of the Impulse Response

The impulse response not only indicates the stability and causality of the system if feedback exists in the system, but also provides information on properties such as the damping, dominating time constant, and time delay. Some of this information, such as the time delay, is useful for parametric model estimation. Therefore, you can use nonparametric impulse response estimation before parametric model estimation to help estimate the parameters. The following example demonstrates how to use the SI Estimate Impulse Response VI to estimate the impulse response and determine the time delay of a system by using the correlation analysis method.

Figure 3-5 shows the front panel of a VI that simulates a system defined by the following equation.

$$y(k) = 0.2u(k-2) + 0.8u(k-3) + 0.3u(k-4)$$

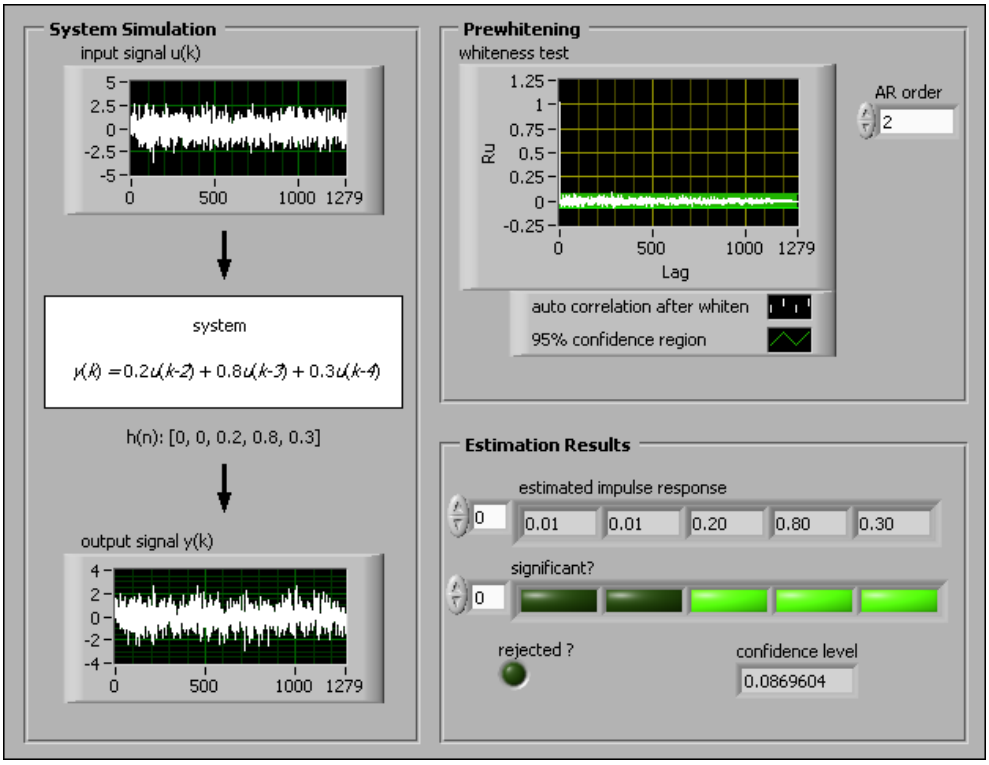

Figure 3-5. Front Panel of Time Delay Example VI

Figure 3-6 shows the block diagram of this VI.

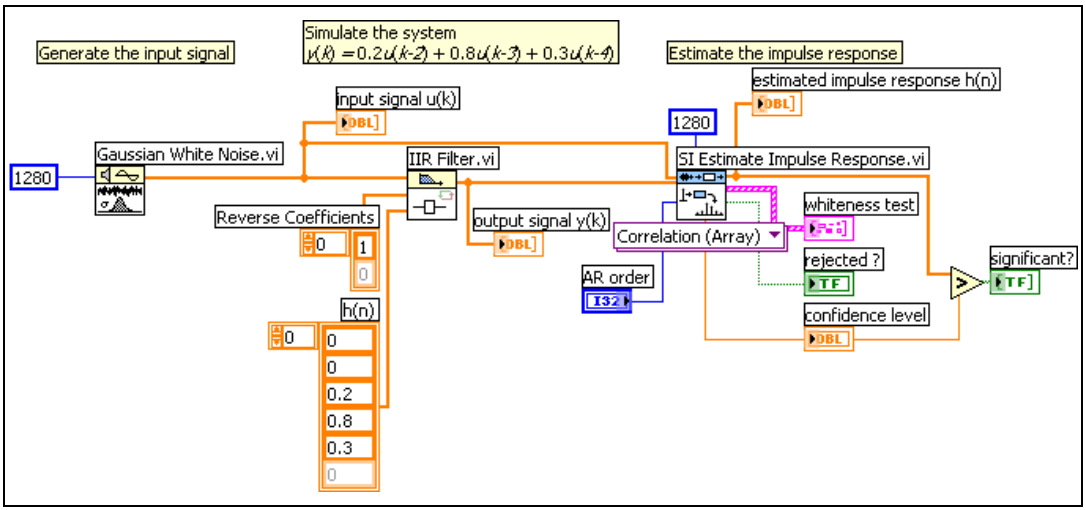

Figure 3-6. Block Diagram of Time Delay Example VI

In Figure 3-5, the two initial values of the **estimated impulse response** are smaller than the **confidence level**. You can have 99.0% confidence that values less than the **confidence level** are insignificant, and you can consider those values to be equal to 0. Therefore, you can conclude that the time delay of the system is 2 because the beginning of the first two values of the impulse response are zero.

Another common application of the impulse response is to detect feedback in systems using the least squares method. If feedback exists in a system, the impulse response of the system becomes significantly large at negative lags and the correlation between the input signal and disturbance $e(k)$ is nonzero. The correlation analysis method assumes the input signal and the disturbance $e(k)$ are independent from each other. Thus, this method cannot estimate accurately the impulse response of the system that contains feedback. Only the least squares method can provide reliable results. Refer to Chapter 6, *Model Estimation Methods in Closed-Loop Systems*, for more information about feedback, closed-loop systems and feedback detection.

Frequency Response

In theory, the results from impulse response estimation and the results from frequency response estimation are equivalent. For example, the Fourier transform of the impulse response $h(n)$, which you can compute using impulse response estimation, equals the frequency response $G(e^{j\omega})$. However, this equivalence does not hold in most real-world applications because of different preprocessing schemes in impulse response estimation and frequency response estimation.

The frequency response provides the complete frequency-domain characteristics of the system, including the passband and the natural frequency of the system. A sinusoidal input signal has the following general form:

$$u(t) = \sin(\omega_0 t)$$

For a linear time-invariant system, the response of a linear time-invariant system to a sinusoidal input also is a sinusoidal signal but potentially with a different magnitude and phase, as shown in the following equation.

$$y(t) = b\sin(\omega_0 t + \theta)$$

where b and θ are the magnitude and phase, respectively, of the frequency response of the system to an input sinusoidal frequency ω_o. If you apply input signals with a number of sinusoids at different frequencies, then you can obtain an estimate of the frequency response $G(\omega)$ of the system at those frequencies. The frequency response is a complex-valued sequence. The magnitude of $G(\omega)$ is the magnitude response of the system and the phase of $G(\omega)$ is the phase response of the system. This method of obtaining the frequency response is straightforward but takes a long time to complete and is sensitive to noise. For these reasons, the System Identification Toolkit uses the spectral analysis method to estimate the frequency response function.

Spectral Analysis Method

You can use the spectral analysis method with any input signal. However, the frequency bandwidth of the input signal must cover the range of interest.

Because the frequency response is the Fourier transform of the impulse response, applying the Fourier transform to both sides of the cross correlation function yields the following equation.

$$\Phi_{uy}(e^{j\omega}) = \Phi_{uu}(e^{j\omega})G(e^{j\omega})$$

$G(e^{j\omega})$ is the frequency response of the system. $\Phi_{uu}(e^{j\omega})$ is the auto-spectral density of the stimulus signal $u(k)$. $\Phi_{uy}(e^{j\omega})$ is the cross-spectral density between the stimulus signal $u(k)$ and the response signal $y(k)$.

You then can use the following equation to compute the frequency response $G(e^{j\omega})$.

$$G(e^{j\omega}) = \frac{\Phi_{uy}(e^{j\omega})}{\Phi_{uu}(e^{j\omega})}$$

You can compute $\Phi_{uu}(e^{j\omega})$ and $\Phi_{uy}(e^{j\omega})$ by applying a fast Fourier transform (FFT) to the autocorrelation R_{uu} and the cross correlation R_{uy}, respectively. As shown in the autocorrelation function R_{uu} and the cross correlation function R_{uy}, described in the *Correlation Analysis* section, the number of data points you need to compute R_{uu} and R_{uy} decreases as the lag τ increases. Therefore, R_{uu} and R_{uy} become inaccurate for a large lag τ.

When computing $\Phi_{uu}(e^{j\omega})$ and $\Phi_{uy}(e^{j\omega})$, you can apply a lag window $w(\tau)$ to R_{uu} and R_{uy} before conducting the FFT operation to improve the accuracy of the frequency response estimation, as shown in the following equations.

$$\Phi_{uu}(e^{j\omega}) = \sum_{\tau=-N}^{N} R_{uu}(\tau) w_m(\tau) e^{-j\omega\tau}$$

$$\Phi_{uy}(e^{j\omega}) = \sum_{\tau=-N}^{N} R_{uy}(\tau) w_m(\tau) e^{-j\omega\tau}$$

The lag window approaches zero when the lag τ is large. The window weighs out the points of R_{uu} and R_{uy} with large lag τ, thereby improving the accuracy of the frequency response estimation. The SI Estimate Frequency Response VI uses a Hanning window as the lag window.

Refer to the book *System Identification Theory for the User*[1] for more information about using a Hanning window.

[1] Ljung, L. 1999. *System Identification Theory for the User*. 2nd ed. Prentice Hall.

Accuracy of the Lag Window

The frequency response with the lag window, $G'(e^{j\omega})$, is equivalent to the moving average version of the frequency response without the lag window, $G(e^{j\omega})$. The average smooths the frequency response, but the smooth frequency response also can deviate more from the true frequency response. Adjusting the length of the lag window can balance the trade-off between variance and bias of the frequency response estimation. The larger the length of the lag window, the fewer points of $G(e^{j\omega})$ the SI Estimate Frequency Response VI averages to compute $G'(e^{j\omega})$, and hence the larger the variance and the smaller the bias of the frequency estimation.

The following example demonstrates how the length of the lag window affects the frequency response estimation. Figure 3-7 shows the front panel of a VI that simulates a system defined by the following equation.

$$y(k) - 1.46y(k-1) + 2.5y(k-2) - 1.46y(k-3) + yk - 4 \;=\; u(k) + 0.45u(k-1) + u(k-2)$$

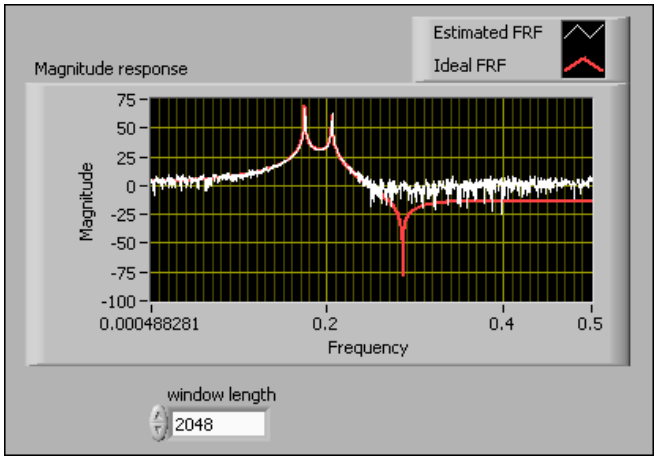

Figure 3-7. Front Panel of Lag Window Example VI

Figure 3-8 shows the block diagram of this VI.

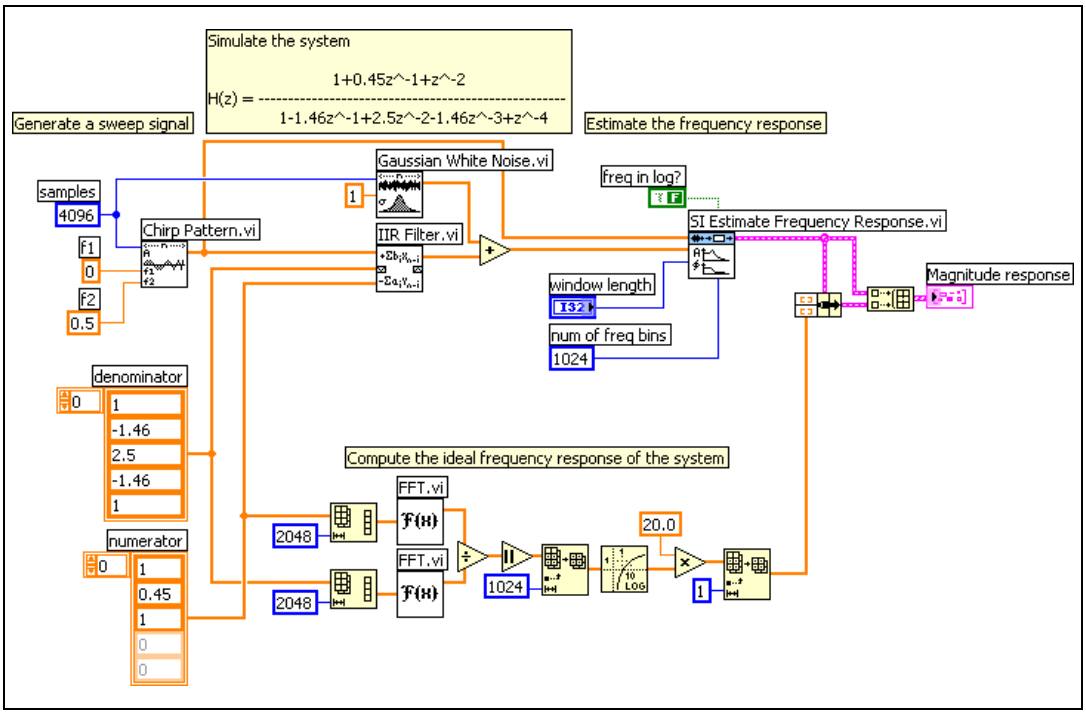

Figure 3-8. Block Diagram of Lag Window Example VI

In this example VI, the input signal $u(k)$ is a swept sine wave whose normalized frequency is from 0 to 0.5. The number of data points in the input signal is 4096. The length of the lag window therefore must be less than or equal to 4096. Figure 3-9 and Figure 3-10 show the resulting frequency responses when the window length is 4096 and 64 respectively.

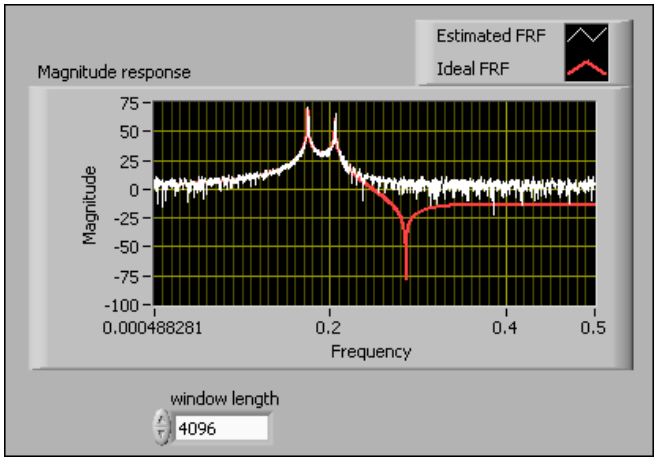

Figure 3-9. Frequency Response with Large Window Length

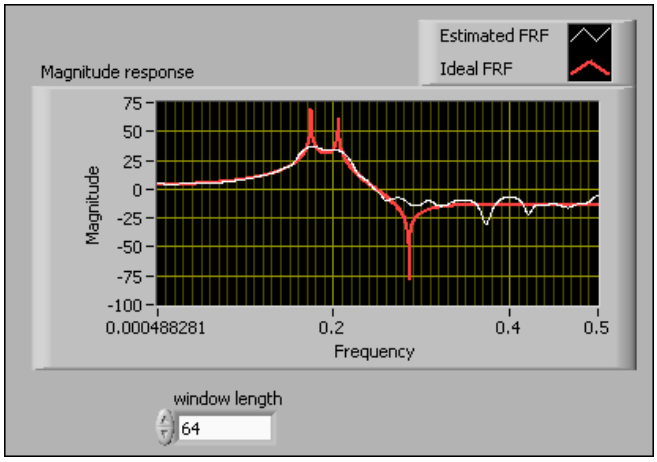

Figure 3-10. Frequency Response with Small Window Length

The frequency response curve is smoother and the variance is smaller when the length of the lag window is small. However, when the length of the lag window is too small, you cannot distinguish between the two close peaks in the frequency response, as shown in Figure 3-10. When the length of the lag window is large, the SI Estimate Frequency Response VI accurately estimates the peaks, as shown in Figure 3-9. The bias is small with a large lag window, but the variance of the estimated frequency response is large with a large lag window.

Setting the length of the lag window to 5–10% of the number of data points when estimating the frequency response often results in a good trade-off between the bias and variance. However, the selection of the length of the lag window is not trivial. The length also depends on the signals, the properties of the system, and the purpose of application. For example, if you want to know the passband of a system, use a smaller lag window. If you want to identify the dynamic properties of a system, such as its natural frequency, use a larger lag window.

Applications of the Frequency Response

The frequency response gives the characteristics of the system in the frequency domain. You can use the frequency response to obtain useful information before applying parametric estimation. For example, you can use the frequency response to determine whether you need to pre-filter the signals or what the model order of the system is. You also can use nonparametric frequency response to verify parametric model estimation results by comparing the frequency response of the parametric model with the nonparametric frequency response.

One example of a real-world application of the frequency response is with the flexible arm, as shown in Figure 3-11. The input of this system is the reaction torque of the structure on the ground. This input is a multi-sine wave with 200 frequency points equally spaced over the frequency band from 0.122 Hz to 24.4 Hz. The output of this system is the acceleration of the flexible arm. The frequency response of this system is not significant outside of the range of interest, which is the frequency band of the input signal, or 0.122 Hz to 24.4 Hz. However, notice that the **magnitude response** has a peak around 42 Hz. The peak around 42 Hz may be the result of noise, or nonlinearity, or another input source. You can use lowpass filtering to remove the 42 Hz peak before applying parametric estimation.

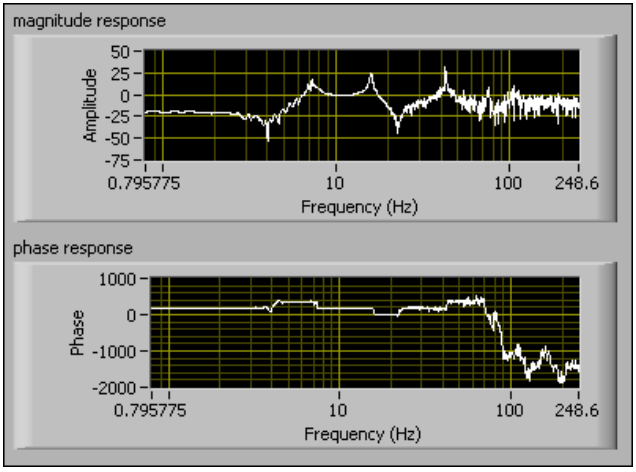

Figure 3-11. Frequency Response of a Flexible Arm

4

Parametric Model Estimation Methods

Parametric models describe systems in terms of difference or differential equations depending on whether a system is represented by a discrete or continuous model. Compared to nonparametric models, parametric models might provide more accurate estimation if users have prior knowledge about the system dynamics to determine model orders, time delays, and so on.

This chapter describes parametric model representations, the assumptions about each estimation method, and reasons for choosing one representation over another.

Parametric Model Representations

The LabVIEW System Identification Toolkit provides four categories of parametric models—general-linear polynomial, transfer function, zero-pole-gain, and state-space. General-linear polynomial, transfer function, and zero-pole-gain models are all polynomial models.

General-Linear Polynomial Model

General-linear polynomial models apply only for discrete systems. Generally, you can describe a discrete system using the following equation, which is known as the general-linear polynomial model.

$$y(k) = z^{-n}G(z^{-1}, \theta)u(k) + H(z^{-1}, \theta)e(k)$$

where $u(k)$ and $y(k)$ are the input and output of the system, respectively

$e(k)$ is the disturbance of the system which usually is zero-mean white noise

$G(z^{-1}, \theta)$ is the transfer function of the deterministic part of the system

$H(z^{-1}, \theta)$ is the transfer function of the stochastic part of the system

The deterministic transfer function specifies the relationship between the output and the input signal. The stochastic transfer function specifies how the random disturbance affects the output signal. Often the deterministic and stochastic parts of a system are referred to as system dynamics and stochastic dynamics, respectively.

The term z^{-1} is the backward shift operator, which is defined by the following equations:

$$z^{-1}x(k) = x(k-1)$$

$$z^{-2}x(k) = x(k-2)$$

$$\cdots$$

$$z^{-\mathrm{n}}x(k) = x(k-\mathrm{n})$$

z^{-n} defines the number of delay samples between the input and the output.

$G(z^{-1}, \theta)$ and $H(z^{-1}, \theta)$ are rational polynomials as defined by the following equations:

$$G(z^{-1}, \theta) = \frac{B(z, \theta)}{A(z, \theta)F(z, \theta)}$$

$$H(z^{-1}, \theta) = \frac{C(z, \theta)}{A(z, \theta)D(z, \theta)}$$

The vector θ is the set of model parameters. Equations in the following sections of this manual do not display θ to make the equations easier to read.

The following equations define $A(z)$, $B(z)$, $C(z)$, $D(z)$, and $F(z)$.

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{k_a} z^{-k_a}$$

$$B(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{k_b - 1} z^{-(k_b - 1)}$$

$$C(z) = 1 + c_1 z^{-1} + c_2 z^{-2} + \ldots + c_{k_c} z^{-k_c}$$

$$D(z) = 1 + d_1 z^{-1} + d_2 z^{-2} + \ldots + d_{k_d} z^{-k_d}$$

$$F(z) = 1 + f_1 z^{-1} + f_2 z^{-2} + \ldots + f_{k_f} z^{-k_f}$$

where k_a, k_b, k_c, k_d, and k_f are the model orders.

The following equation describes a general-linear polynomial model.

$$A(z)y(k) = \frac{z^{-n}B(z)}{F(z)}u(k) + \frac{C(z)}{D(z)}e(k) = \frac{B(z)}{F(z)}u(k-n) + \frac{C(z)}{D(z)}e(k)$$

Figure 4-1 depicts the signal flow of a general-linear polynomial model.

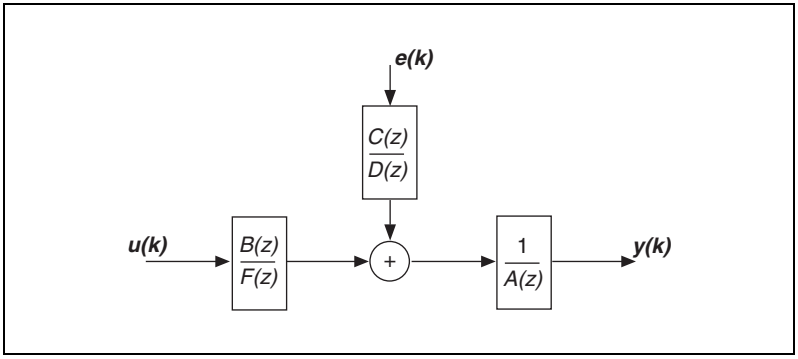

Figure 4-1. Signal Flow of a General-Linear Polynomial Model

A general-linear polynomial model provides flexibility for both system dynamics and stochastic dynamics. You can use the SI Estimate General Linear Model VI to estimate general-linear polynomial models.

Setting one or more of $A(z)$, $C(z)$, $D(z)$, and $F(z)$ equal to 1 can create simpler models such as autoregressive with exogenous terms (ARX), autoregressive-moving average with exogenous terms (ARMAX), output-error, and Box-Jenkins models, which you commonly use in real-world applications.

ARX Model

When $C(z)$, $D(z)$, and $F(z)$ equal 1, the general-linear polynomial model reduces to an ARX model. The following equation describes an ARX model.

$$A(z)y(k) = z^{-n}B(z)u(k) + e(k) = B(z)u(k-n) + e(k)$$

Note The backward shift operator makes $z^{-n}u(k) = u(k-n)$.

Figure 4-2 depicts the signal flow of an ARX model.

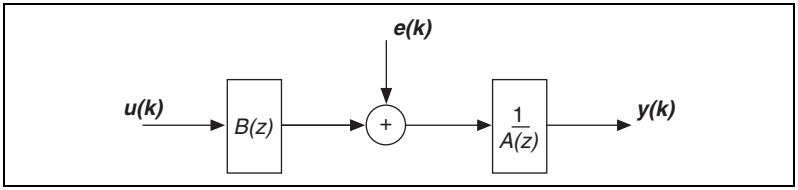

Figure 4-2. Signal Flow of an ARX Model

The ARX model is the simplest model incorporating the stimulus signal. However, the ARX model captures some of the stochastic dynamics as part of the system dynamics. In this model, the transfer function of the deterministic part $G(z^{-1}, \theta)$ of the system and the transfer function of the stochastic part $H(z^{-1}, \theta)$ of the system have the same set of poles. This coupling can be unrealistic. The system dynamics and stochastic dynamics of a system do not share the same set of poles all the time. You can reduce this disadvantage if the signal-to-noise ratio is high.

When the disturbance $e(k)$ of a system is not white noise, the coupling between the deterministic and stochastic dynamics can bias the estimation of the ARX model. You can set the model order higher than the actual model order to minimize the estimation error, especially when the signal-to-noise ratio is low. However, increasing the model order can change some dynamic characteristics of the model, such as the stability of the model.

The identification method for the ARX model is the least squares method, which is a special case of the prediction error method. The least squares method is the most efficient polynomial estimation method because this method solves linear regression equations in analytic form. Moreover, the solution is unique. Refer to the *LabVIEW System Identification Toolkit Algorithm References* manual (`SIreference.pdf`), available in the `labview\manuals` directory, for more information about the least squares and prediction error methods.

You can use the SI Estimate ARX Model VI to estimate ARX models.

ARMAX Model

When $D(z)$ and $F(z)$ equal 1, the general-linear polynomial model reduces to the ARMAX model. The following equation describes an ARMAX model.

$$A(z)y(k) = z^{-n}B(z)u(k) + C(z)e(k) = B(z)u(k - n) + C(z)e(k)$$

Figure 4-3 depicts the signal flow of an ARMAX model.

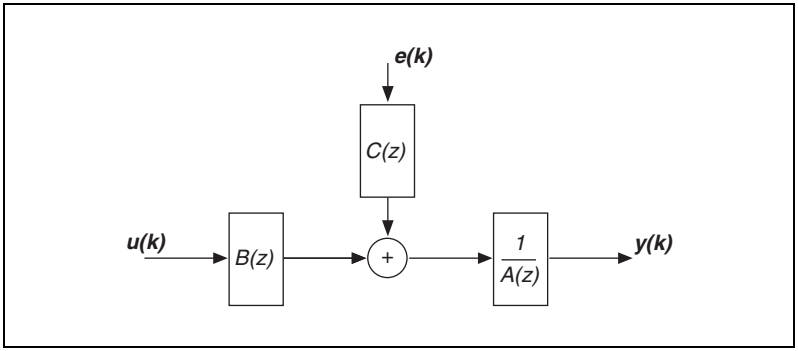

Figure 4-3. Signal Flow of an ARMAX Model

Unlike the ARX model, the system structure of an ARMAX model includes the stochastic dynamics. ARMAX models are useful when you have dominating disturbances that enter early in the process, such as at the input. For example, a wind gust affecting an aircraft is a dominating disturbance early in the process. The ARMAX model has more flexibility than the ARX model in handling models that contain disturbances.

The identification method of the ARMAX model is the prediction error method. The SI Estimate ARMAX Model VI uses the Gauss-Newton method to optimize the mean square value of the prediction error when searching for the optimal ARMAX model. This searching process is iterative and might converge to a local minimum. Therefore, you must validate the estimated model. If the estimated model passes the validation test, you can use this model even if the SI Estimate ARMAX Model VI might locate only a local minimum. Refer to Chapter 5, *Partially Known Model Estimation Methods*, for more information about optimization and local minima.

Output-Error Model

When $A(z)$, $C(z)$, and $D(z)$ equal 1, the general-linear polynomial model reduces to the output-error model. The following equation describes an output-error model.

$$y(k) = \frac{z^{-n}B(z)}{F(z)}u(k) + e(k) = \frac{B(z)}{F(z)}u(k-n) + e(k)$$

Figure 4-4 depicts the signal flow of an output-error model.

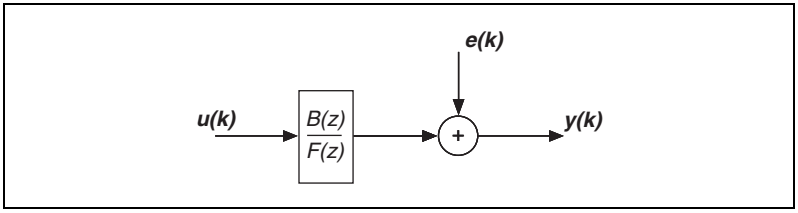

Figure 4-4. Signal Flow of an Output-Error Model

The output-error model describes the system dynamics separately from the stochastic dynamics. The output-error model does not use any parameters for simulating the disturbance characteristics.

The identification method of the output-error model is the prediction error method, which is the same as that of the ARMAX model. If the disturbance $e(k)$ is white noise, all minima are global. However, a local minimum can exist if the disturbance is not white noise. You can use the SI Estimate OE Model VI to estimate output-error models.

Box-Jenkins Model

When $A(z)$ equals 1, the general-linear polynomial model reduces to the Box-Jenkins model. The following equation describes a Box-Jenkins model.

$$y(k) = \frac{z^{-n}B(z)}{F(z)}u(k) + \frac{C(z)}{D(z)}e(k) = \frac{B(z)}{F(z)}u(k-n) + \frac{C(z)}{D(z)}e(k)$$

Figure 4-5 depicts the signal flow of a Box-Jenkins model.

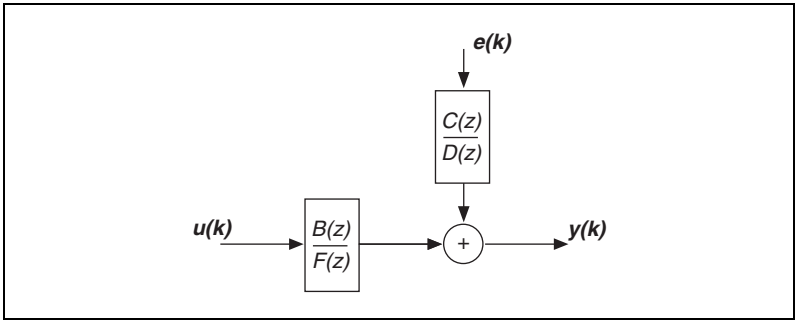

Figure 4-5. Signal Flow of a Box-Jenkins Model

The Box-Jenkins model provides a complete model of a system. The Box-Jenkins model represents disturbance properties separately from system dynamics. This model is useful when you have disturbances that enter late in the process, such as measurement noise on the output.

The identification method of the Box-Jenkins model is the prediction error method, which is the same as that of the ARMAX model. You can use the SI Estimate BJ Model VI to estimate Box-Jenkins models.

AR Model

When $C(z)$, $D(z)$, and $F(z)$ equal 1 and $B(z)$ equals 0, the general-linear polynomial model reduces to the AR model. The following equation describes an AR model.

$$A(z)y(k) = e(k)$$

Figure 4-6 depicts the signal flow of an AR model.

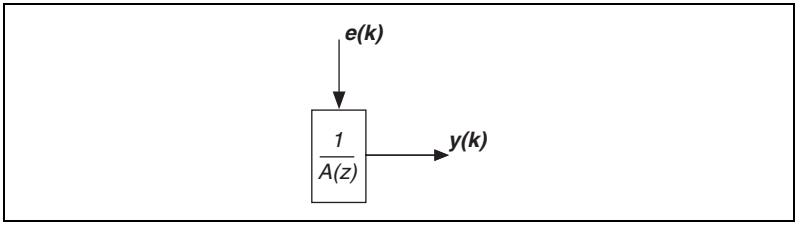

Figure 4-6. Signal Flow of an AR Model

The AR model does not include the dynamics between the input and output. Therefore, the AR model is more suitable for representing signals rather than a system because a system generally has an input and an output.

Time series analysis methods, such as power spectrum envelope estimation, prewhitening, and linear prediction coding, commonly use the AR model. Refer to the *Time Series Analysis Tools User Manual* at ni.com/manuals for more information about time series analysis methods.

If you consider $A(z)$ to be a filter, $A(z)y(k)$ is the filtering of $A(z)$ on the signal $y(k)$. The result of the filtering is white noise $e(k)$, as shown in the AR model equation. Hence, the filter $A(z)$ also is known as the prewhitening filter. From the frequency-domain standpoint, the prewhitening filter $A(z)$ suppresses the spectrum at frequencies where the magnitude of the spectrum is large. Suppressing the high-magnitude frequencies results in a flat spectrum. Refer to Chapter 3, *Nonparametric*

Model Estimation Methods, for examples of using the AR model for prewhitening.

As shown in the AR model equation, if you know the AR coefficients $A(z)$ and the noise $e(k)$, you can reconstruct the signal $y(k)$. $A(z)$ and $e(k)$ completely characterize a signal. $A(z)$ normally has a small number of coefficients. $e(k)$ has a small dynamic range and requires a smaller number of bits for encoding. Therefore, you can use the AR model for compression purposes in a process known as linear prediction coding (LPC). Speech and vibration signal processing methods, such as compression and pattern recognition, commonly use LPC. You also can use $A(z)$ and $e(k)$ to estimate the power spectrum of the signal $y(k)$. Use the SI Estimate AR Model VI to estimate AR models.

Transfer Function Model

You can use a transfer function to define either a continuous system or a discrete system using the following equations, respectively.

$$y(t) = G(s)u(t) + e(t)$$

$$y(k) = G(z)u(k) + e(k)$$

Let the parameters of the model be the numerator and denominator coefficients of the transfer function. The transfer function provides a mathematical representation of the relationship between one input and one output. The following equations define the continuous and discrete transfer functions where the numerator and denominator are polynomials.

Continuous Transfer Function Model

$$G(s) = \frac{b_0 + b_1 s + \ldots + b_{m-1} s^{m-1} + b_m s^m}{a_0 + a_1 s + \ldots + a_{n-1} s^{n-1} + a_n s^n}$$

Discrete Transfer Function Model

$$G(z) = \frac{b_0 + b_1 z + \ldots + b_{m-1} z^{m-1} + b_m z^m}{a_0 + a_1 z + \ldots + a_{n-1} z^{n-1} + a_n z^n}$$

You can use the SI Estimate Transfer Function Model VI to estimate both continuous and discrete models. For discrete models, this VI implements the prediction error method. For continuous models, this VI internally

performs the following three consecutive steps to estimate the model. The SI Estimate Transfer Function Model VI first calculates a discrete model with the prediction error method. This VI then applies the Zero-Order-Hold method to convert the discrete model to a continuous model. Finally, this VI uses the Gauss-Newton method to optimize the continuous model this VI converted in the second step. Refer to the *LabVIEW Control Design Toolkit User Manual,* available at ni.com/manuals, for more information about the Zero-Order-Hold method.

Transfer function models describe only the deterministic part of the system. For stochastic control, general-linear polynomial models commonly are used because these models separately describe the deterministic and stochastic parts of a system. However, in classical control engineering, the deterministic part of the system is more important than the stochastic part. Therefore, you can take advantage of the relationship between input and output signals of the transfer function model to describe the deterministic part of the system.

Zero-Pole-Gain Model

If you rewrite the equations for the transfer function model to show the locations of the zeroes and poles of the dynamic system, you obtain the zero-pole-gain model. The following equations represent the continuous and discrete zero-pole-gain models, respectively.

$$G(s) = \frac{K(s-z_1)(s-z_2)\ldots(s-z_m)}{(s-p_1)(s-p_2)\ldots(s-p_n)}$$

$$G(z) = \frac{K(z-z_1)(z-z_2)\ldots(z-z_m)}{(z-p_1)(z-p_2)\ldots(s-p_n)}$$

where K is the gain, z_i are the zeroes, and p_j are the poles.

The System Identification Toolkit does not provide a VI to estimate zero-pole-gain models directly because you can use the SI Model Conversion VI to convert another model representation to a zero-pole-gain model. Refer to Chapter 8, *Analyzing, Validating, and Converting Models*, for more information about converting other model representations to zero-pole-gain models. Refer to the *LabVIEW Help,* available by selecting **Help»Search the LabVIEW Help**, for more information about the SI Model Conversion VI.

State-Space Model

In addition to polynomial models, you also can estimate state-space models with the System Identification Toolkit. The state-space model describes a system using difference or differential equations with an auxiliary state vector. The following equations describe a discrete state-space model.

$$x(k+1) = Ax(k) + Bu(k) + Ke(k)$$

$$y(k) = Cx(k) + Du(k) + e(k)$$

The following equations describe a continuous state-space model.

$$\dot{x} = Ax(t) + Bu(t) + Ke(t)$$

$$y = Cx(t) + Du(t) + e(t)$$

where x is the state vector

k is the model sampling time multiplied by the discrete time step, where the discrete time step equals 0, 1, 2, ...

t is the time for the continuous model

A is the system matrix that describes the dynamics of the states of the system

B is the input matrix that relates the inputs to the states

C is the output matrix that relates the outputs to the states

D is the transmission matrix that relates the inputs to the outputs

K is the Kalman gain

The dimension of the state vector x is the only setting you need to provide for the state-space model. The state-space transfer matrices, A, B, C, and D, often reflect physical characteristics of a system.

The state-space model is the most convenient model in describing multivariable systems. State-space models often are preferable to polynomial models, especially in modern control applications that focus on multivariable systems.

The System Identification Toolkit provides the SI Estimate State-Space Model VI to estimate discrete state-space models using two methods—the deterministic-stochastic subspace and the realization methods. The deterministic-stochastic subspace method uses principal component analysis to estimate parameters. This method uses both stimulus and

response signals to estimate state-space models. The realization method uses the impulse response to estimate only the deterministic state-space model. This method does not include stochastic parts of the system in the model structure. Therefore, the difference between these two methods is that the deterministic-stochastic subspace method includes noise in the model structure, whereas the realization method does not. Refer to the *LabVIEW System Identification Toolkit Algorithm References* manual (`SIreference.pdf`), available in the `labview\manuals` directory, for more information about the deterministic-stochastic subspace method and the realization method.

For continuous state-space models, the System Identification Toolkit supports partially known model estimation methods. You must provide an initial guess for each parameter before conducting estimation. Refer to Chapter 5, *Partially Known Model Estimation Methods*, for more information about partially known, continuous, state-space model estimation methods.

User-Defined Model

When the model you want to estimate does not fall into the above four categories, you can define your own model by revising a template VI. You can find template VIs in the `labview\vi.lib\addons\System Identification\User-Defined Model Templates.llb` directory. Then you can estimate the model you define using the SI Estimate User-Defined Model VI. The SI Estimate User-Defined Model VI enables you to estimate some other model representations in addition to the general-linear polynomial, transfer function, zero-pole-gain, and state-space models that the System Identification Toolkit directly supports. For example, you can use this VI to estimate nonlinear models. With this VI, you also can estimate your own linear models that you define early.

Refer to the Estimate Hammerstein Model example VI, the Estimate Hammerstein-Wiener Model example VI, the Estimate Wiener Model example VI, and the Parameterize Nonlinear Differential Equation example VI for more information about using the SI Estimate User-Defined Model VI to estimate nonlinear models. You can access these example VIs by selecting **Help»Find Examples** to display the NI Example Finder and then navigating to the **Toolkits and Modules»System Identification** folder. Refer to the *LabVIEW Help* for more information about the SI Estimate User-Defined Model VI.

Polynomial Models versus State-Space Models

Selecting the correct model type and model order is crucial for successfully estimating a parametric model. In general, state-space models provide a more complete representation of the system, especially for multiple-input multiple-output (MIMO) systems, than polynomial models because state-space models are similar to first principle models that can provide more degree of freedom in describing MIMO systems.

The identification procedure for state-space models does not involve nonlinear optimization so the estimation reaches a solution regardless of the initial guess. Moreover, the parameter settings for the state-space model are simpler. You need to select only the order, or the number of states, of the model. The order can come from prior knowledge of the system. You also can determine the order by analyzing the singular values of the information matrix. However, the states that the state-space identification procedure identifies might not reflect the physical characteristics of a system accurately. Using a similarity transformation, you can identify equivalent models with states that better represent the system. Similarity transformations enable you to transform the states without misrepresenting the input-output behavior of the system. Refer to the book *Linear systems*[1] for more information about similarity transforms. Refer to Chapter 9, *System Identification Case Study*, for an example of how to estimate the order of the state-space model of a system.

When model order is high, state-space models are preferable to polynomial models. Polynomial models with high order might encounter numerical problems in computation.

Determining Parameters for the Prediction Error Method

The identification method for most of the polynomial models is the prediction error method. Determining the delay and model order for the prediction error method is typically a trial-and-error process. The following steps can help you obtain a suitable model. These steps are not the only methods you can use, nor are these steps a comprehensive procedure.

1. Obtain useful information about the model order by observing the number of resonance peaks in the nonparametric frequency response function. Normally, the number of peaks in the magnitude response equals half the order of $A(z)F(z)$.

[1] Kailath, T. 1980. *Linear systems*. Prentice Hall.

2. Obtain a reasonable estimate of the delay by observing the impulse response or by testing reasonable values in a medium-sized ARX model. Choose the delay that provides the best model fit based on prediction errors or another criterion.
3. Test various ARX model orders with this delay, choosing those orders that provide the best fit.
4. Reduce the model order by plotting the poles and zeros with confidence intervals and looking for potential cancellations of pole-zero pairs. The resulting model might be unnecessarily high in order because the ARX model describes both the system dynamics and noise properties using the same set of poles. The ARMAX, output-error, and Box-Jenkins models use the resulting orders of the poles and zeros as the *B* and *F* model parameters and the first- or second-order models for the noise characteristics. Refer to Chapter 8, *Analyzing, Validating, and Converting Models*, for more information about pole-zero plot and confidence intervals.
5. Determine if additional signals influence the output if you cannot obtain a suitable model at this point. You can incorporate measurements of these signals as extra input signals.

If you still cannot obtain a suitable model, additional physical insight into the problem might be necessary. Compensating for nonlinear sensors or actuators and handling important physical nonlinearities often are necessary in addition to using a ready-made model.

From the prediction error standpoint, the higher the order of the model is, the better the model fits the data because the model has more degrees of freedom. However, you need more computation time and memory for higher orders. The parsimony principle says to choose the model with the smallest degree of freedom, or number of parameters, if all the models fit the data well and pass the verification test. The criteria to assess the model order therefore not only must rely on the prediction error but also must incorporate a penalty when the order increases. Akaike's Information Criterion (AIC), Akaike's Final Prediction Error Criterion (FPE), and the Minimum Description Length Criterion (MDL) are criteria you can use to estimate the model order. The SI Estimate Orders of System Model VI implements the AIC, FPE, and MDL methods to search for the optimal model order in the range of interest. You also can plot the prediction error as a function of the model dimension and then visually find the minimum in the curve or apply an F-test to obtain an appropriate estimation of the model order.

Akaike's Information Criterion

The Akaike's Information Criterion (AIC) is a weighted estimation error based on the unexplained variation of a given time series with a penalty term when exceeding the optimal number of parameters to represent the system. For the AIC, an optimal model is the one that minimizes the following equation:

$$AIC = V_n\left(1 + \frac{2p}{N}\right)$$

N is the number of data points, V_n is an index related to the prediction error, or the residual sum of squares, and p defines the number of parameters in the model.

Final Prediction Error Criterion

The Final Prediction Error Criterion (FPE) estimates the model-fitting error when you use the model to predict new outputs. For the FPE, an optimal model is the one that minimizes the following equation:

$$FPE = V_n\left(1 + \frac{2p}{N-p}\right)$$

You want to choose a model that minimizes the FPE, which represents a balance between the number of parameters and the explained variation.

Minimum Description Length Criterion

The Minimal Description Length Criterion (MDL) is based on V_n plus a penalty for the number of terms used. For the MDL, an optimal model is the one that minimizes the following equation:

$$MDL = V_n\left(1 + \frac{p\ln N}{N}\right)$$

You want to choose a model that minimizes the MDL, which allows the shortest description of data you measure.

5

Partially Known Model Estimation Methods

The model estimation methods introduced in previous chapters assume that all model parameters are adjustable. However, in many real-world applications, you cannot adjust all the parameters arbitrarily, because the parameters might have constraints. For example, in some chemical processes, water must flow only in one direction. When estimating the flow rate of water, you know that the flow rate cannot be negative. Thus, the constraint is that the flow rate must be a positive value. You must consider this constraint and any other constraints when you estimate the flow rate of water in this process. Models for which you can set constraints on the parameters are called partially known models.

This chapter compares unknown and partially known model estimation methods, describes the methods you can use to define partially known models, and explains how user-defined initial values affect estimation results. This chapter also provides a case study to estimate a partially known model.

Comparing Unknown and Partially Known Model Estimation Methods

When you use the model estimation methods described in previous chapters, you assume systems are unknown. However, in practice, many systems are partially known because you have information about the underlying dynamics or some of the physical parameters. Unknown and partially known models also are known as black-box and grey-box models, respectively.

Conventional black-box model estimation methods do not use the prior knowledge you have about a model. When estimating a black-box model, you cannot set constraints on parameters. You can set only the model order that specifies the number of parameters to calculate. With black-box estimation methods, you use either an algorithm or a trial-and-error method to vary model parameters until the behavior of the model matches the

measured input-output data. You can use the estimated parameters to reproduce the response of the system accurately. However, the parameters might not have any physical meanings.

With grey-box model estimation methods, you can incorporate prior knowledge of a system by setting constraints on certain parameters. These constraints reflect the knowledge you have of the physical system. You therefore can obtain a more realistic parameter estimation. Such constraints usually follow one of the following guidelines:

- A parameter must be as close to a value as possible.
- A parameter must be between two values.
- Two or more parameters must correlate to each other.

With these constraints, you can specify the system model more accurately. Parameter constraints increase the possibility of the System Identification VIs locating the optimal parameters that describe the real-world model. Parameter constraints also improve the accuracy of locating these optimal parameters.

Defining and Estimating Partially Known Models

Before estimating partially known models, you first must define those models. Using prior knowledge, you choose a model for the plant in a system and set parameter constraints for the model. You then can estimate the model to represent the real-world plant. The LabVIEW System Identification Toolkit provides two VIs with which you can define partially known models—the SI Create Partially Known State-Space Model VI and the SI Create Partially Known Continuous Transfer Function Model VI.

The SI Create Partially Known State-Space Model VI enables you to define partially known continuous or discrete state-space models. Refer to the *State-Space Model* section of Chapter 4, *Parametric Model Estimation Methods*, for more information about state-space model definitions.

You can use the SI Create Partially Known State-Space Model VI, for example, to define a state-space model that represents an RLC circuit consisting of a resistor *R*, an inductor *L*, and a capacitor *C*. Using prior knowledge, you describe the relationship of *R*, *L*, and *C* with the following equations:

$$A = \begin{bmatrix} 0 & 1 \\ -1/(L \times C) & -R/L \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1/(L \times C) \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$D = 0$$

You also can use prior knowledge to define the initial guesses and upper and lower limits of *R*, *L*, and *C*. The SI Create Partially Known State-Space Model VI uses variables rather than numerical values to construct a symbolic model. As Figure 5-1 shows, you use variable names, such as *R*, *L*, and *C*, in the **symbolic A**, **symbolic B**, **symbolic C**, **and symbolic D** inputs to define the RLC circuit. Then you specify values for *R*, *L*, and *C* in the **variables** input.

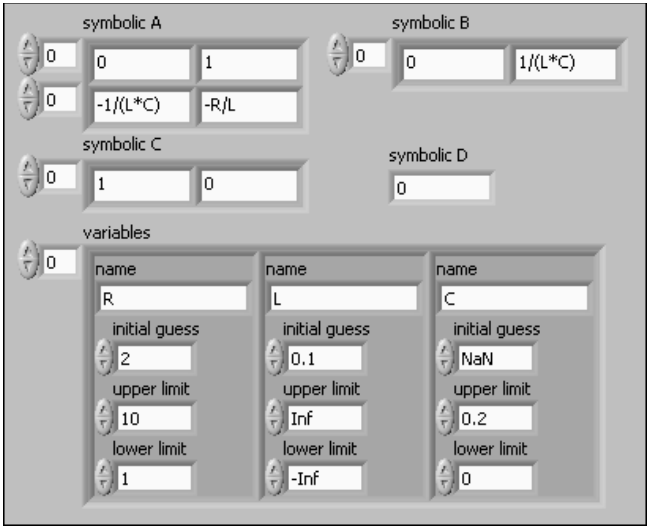

Figure 5-1. Using Variables to Create a Symbolic State-Space Model

The SI Create Partially Known Continuous Transfer Function Model VI enables you to define partially known continuous transfer function models. The following equation represents a continuous transfer function model.

$$\frac{K_p(1 + a_1 s + a_2 s^2 \ldots)}{\prod_i (1 + T_p^i s) \prod_j (1 + 2\ r^j s / w^j + (s/w^j)^2)} e^{-sT_d}$$

where K_p is the transfer function gain

T_d is the delay

T_p is the first-order time constant

w is the natural frequency

r is the damping ratio

s represents the time

You can apply the prior knowledge you have about the parameters K, T_d, T_p, w, and r to the **static gain**, **delay(s)**, **T_p(s)**, **natural freq (rad/s)**, and **damping ratio** inputs, respectively, of the SI Create Partially Known Continuous Transfer Function Model VI by defining the initial guesses and upper and lower limits.

Refer to the *Transfer Function Model* section of Chapter 4, *Parametric Model Estimation Methods*, for more information about continuous transfer function models.

With the System Identification Toolkit and a partially known model, you can set constraints on each parameter of a state-space or continuous transfer function model in two ways—with an upper and lower limit or with an initial guess.

Setting Parameter Constraints with a Range

If you have prior knowledge of a parameter, you can set constraints by providing upper and lower limits for the parameter. With the limit range, the SI Estimate Partially Known State-Space Model VI randomly selects a value within the range as an initial guess of the parameter. From this initial value, the VI then performs optimization to minimize the difference between the estimated output and the measured real output. The goal of constraint optimization is to find a global optimum, or the smallest

difference between the estimated output and the real output, with parameters of physical meaning. Successfully finding the global optimum depends on the limit range you set and the random initial value the SI Estimate Partially Known State-Space Model VI selects.

To increase the possibility of finding the global optimum, complete the following steps:

1. Use prior knowledge to set the range as narrow as possible.
2. Perform multiple estimates with the range you set. You might get different optimization results because the SI Estimate Partially Known State-Space Model VI randomly selects an initial value within the range each time you run the VI. If you repeatedly obtain the same result, this result might be the optimum you want to find. If you obtain inconsistent results, either choose the result that best meets the system requirements, or continue with step 3 to adjust the limit range.
3. Select one of the previous results you got in step 2 according to the prior knowledge you have of the system. Narrow the range in which the result falls. Run the SI Estimate Partially Known State-Space Model VI multiple times. A consistent result you get might be the optimum you want to find. Otherwise, repeat this step until you find a consistent result.

You set limits in the SI Estimate Partially Known Continuous Transfer Function Model VI the same way you do in the SI Estimate Partially Known State-Space Model VI.

Setting Parameter Constraints with an Initial Guess

If you have information about a certain parameter and can estimate a value for that parameter, you can refine estimation by using that value as an initial guess.

The SI Estimate Partially Known State-Space Model VI and the SI Estimate Partially Known Continuous Transfer Function Model VI perform optimization using the initial guess you provide. These two VIs then use the upper and lower limit settings you specify as boundary constraints during the optimization process.

The initial guess you provide greatly affects the performance of any optimization technique. Whether an optimization process reaches a global optimum depends on the initial guess. With some initial guesses, optimization processes might locate only a local optimum, which is the smallest difference between the estimated output and the real output within a certain smaller range rather than in the whole range of interest. Therefore,

to decrease the risk of locating a local optimum instead of the global optimum, try different initial guesses. Figure 5-2 shows an example of different estimations resulting from different initial guesses and illustrates the importance of setting different initial guesses to find the global optimum.

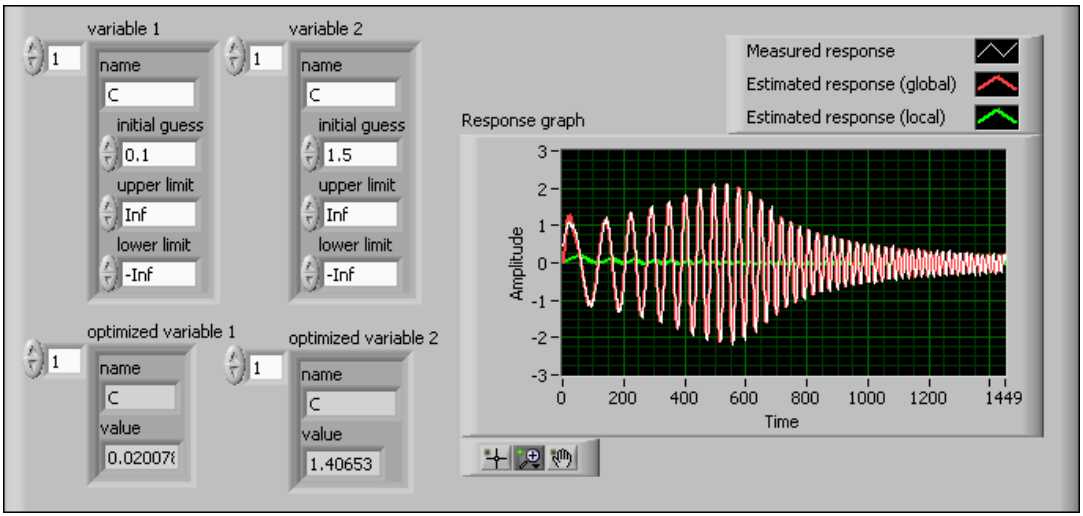

Figure 5-2. Estimating a Local Optimum and a Global Optimum from Different Initial Guesses

As Figure 5-2 shows, if you set *C* to an initial guess of 0.1, you obtain an optimized value of 0.02. You can see the **Estimated response (global)** plot and the **Measured response** plot match in the **Response graph**. This response from the estimated model is close to the real-world model response. However, if you set the initial guess of *C* to 1.5, you get an optimized value of 1.41. The **Estimated response (local)** plot does not match the **Measured response** plot in the **Response graph**. Thus, with this initial guess, the estimated model response does not represent the real-world model response accurately.

Partially Known Model Estimation Case Study

This section contains an example that uses the prior knowledge you have about a system to define and estimate state-space models. You use the same procedure when estimating continuous transfer function models. However, you apply different methods to define continuous transfer function models. Refer to the *Defining and Estimating Partially Known Models* section of this chapter for more information about defining transfer function models.

Figure 5-3 shows an RLC circuit, where u is the input voltage, y is the output voltage, i_L is the current, and u_C is the capacitor voltage. In this example, y equals the capacitor voltage u_C.

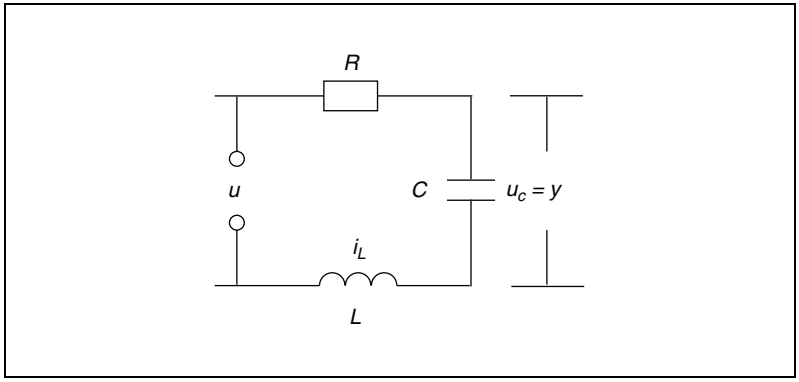

Figure 5-3. An RLC Circuit Representing a State-Space Model

Suppose R is 1.5 Ω and L and C are unknown. You can complete the following steps to identify the values of L and C.

1. Apply a wide-band voltage to u and measure the output y simultaneously. The Continuous State-Space Model of an RLC Circuit example VI uses a chirp signal from 0.5 Hz to 6 Hz as the stimulus signal. The response to the chirp signal is the response signal. This example VI then preprocesses the stimulus and response signals to remove the offset level in these signals.
2. Define a model for this circuit. Because you have information about the approximate values of L and C, you can build a partially known state-space model or a partially known transfer function model.
3. Estimate the model you defined in step 2 and then estimate L and C.

The Continuous State-Space Model of an RLC Circuit example VI guides you through defining and estimating a state-space model for the RLC circuit. You can access this example VI by selecting **Help»Find Examples** to display the NI Example Finder and then navigating to the **Toolkits and Modules»System Identification** folder. Refer to the NI Developer Zone at `ni.com/zone` for an example of using a partially known transfer function model to estimate the RLC circuit.

You can use the following first-order differential equation to represent the relationship between the capacitor voltage and the current of this RLC circuit.

$$\dot{u_C} = \frac{1}{C} i_L \tag{5-1}$$

You can use the following first-order differential equation to represent the voltage relationship in this RLC circuit.

$$R i_L + u_C + L \dot{i_L} = u \tag{5-2}$$

By manipulating Equations 5-1 and 5-2, you can deduce the continuous state-space model for this RLC circuit using the following two equations:

$$\begin{bmatrix} \dot{u_C} \\ \dot{i_L} \end{bmatrix} = \begin{bmatrix} 0 & 1/C \\ -1/L & -R/L \end{bmatrix} \begin{bmatrix} u_C \\ i_L \end{bmatrix} + \begin{bmatrix} 0 \\ 1/L \end{bmatrix} u$$

$$y = u_C = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} u_C \\ i_L \end{bmatrix}$$

The System Identification Toolkit provides the SI Create Partially Known State-Space Model VI with which you can build the symbolic state-space model for this circuit, as shown in Figure 5-4.

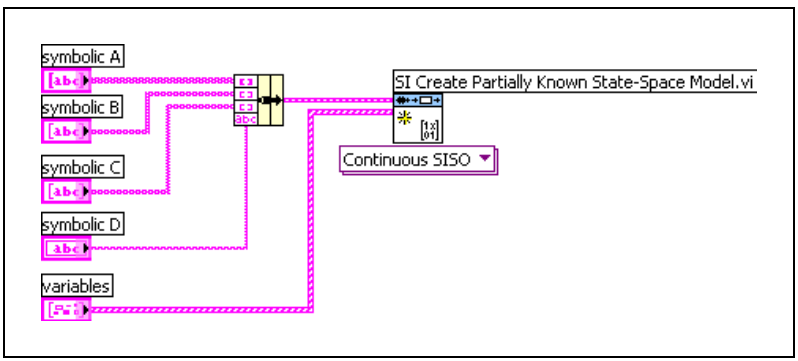

Figure 5-4. Defining a Partially Known State-Space Model

You specify the symbolic state-space model using formula strings, such as `1/C`, `-1/L`, and `-1.5/L`, with *L* and *C* as variables. Then you define *L* and

C with the **variables** input, as shown in Figure 5-5. Using prior knowledge, you know that *L* is a positive value around the initial value of 0.1 H, and *C* is a value between 0 F and 0.3 F.

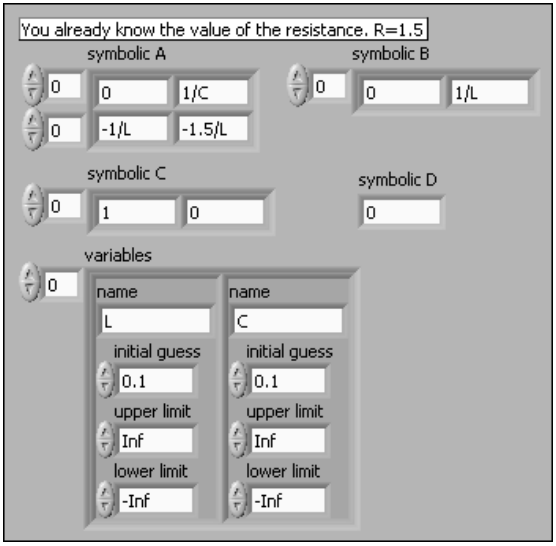

Figure 5-5. Setting Model Parameters of the RLC circuit

Next, you can estimate the state-space model with the SI Estimate Partially Known State-Space Model VI, as shown in Figure 5-6.

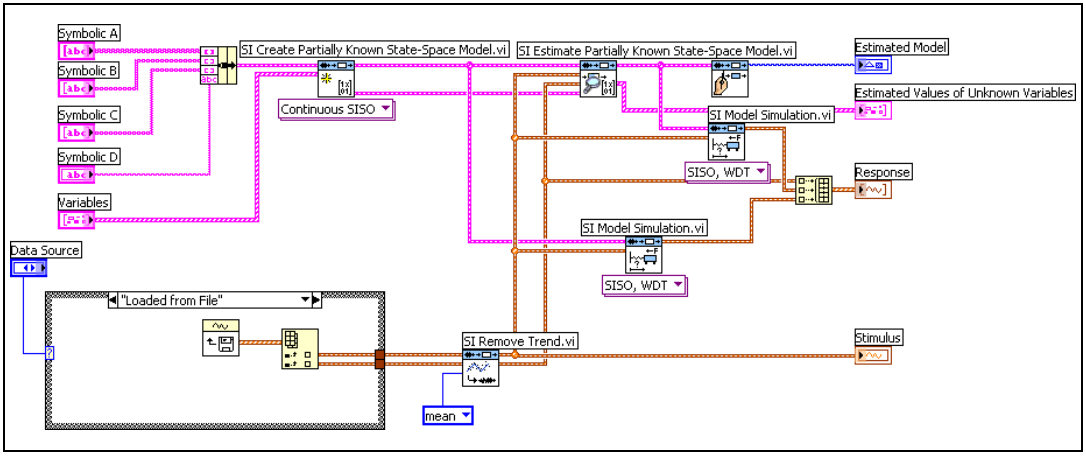

Figure 5-6. Estimating a Partially Known State-Space Model

The SI Estimate Partially Known State-Space Model VI estimates each parameter of the model. You obtain the estimated model and optimized variables of the model after this VI performs an optimization. In this example, you obtain the values 0.20 H for *L* and 0.02 F for *C,* as shown in Figure 5-7.

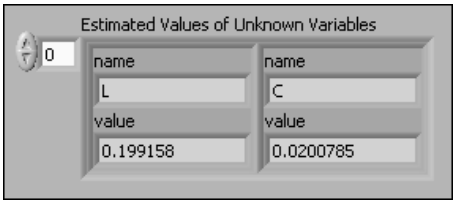

Figure 5-7. Optimized Variables of the RLC Circuit

The Continuous State-Space Model of an RLC Circuit example VI uses the SI Draw Model VI and the values of *L* and *C* you obtain to display the estimated model in a picture indicator, as shown in Figure 5-8.

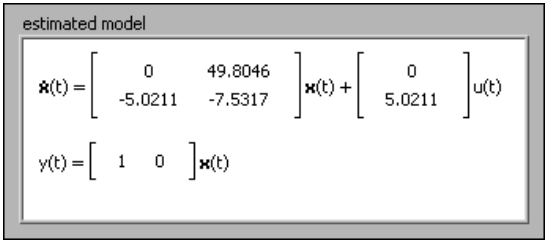

Figure 5-8. The Estimated Model of the RLC Circuit

You then can determine how accurately this model simulates the real-world plant by validating the model. Refer to Chapter 4, *Parametric Model Estimation Methods*, for more information about validating models. Refer to Chapter 9, *System Identification Case Study*, for an example of validating a model.

You can select **Help»Find Examples** to display the NI Example Finder and then navigate to the **Toolkits and Modules»System Identification** book to view more examples of using the System Identification Toolkit to estimate the partially known model of a plant.

6

Model Estimation Methods in Closed-Loop Systems

Unlike the systems discussed in the previous chapters, systems in many real-world applications contain feedback. Feedback is a process in which the output signal of a plant is passed, or fed back, to the input to regulate the next output. Systems without feedback are open-loop systems. Systems with feedback are closed-loop systems.

In an open-loop system, the stimulus signal and the output noise do not correlate with each other. In a closed-loop system, the stimulus signal correlates to the output noise. Though you can apply many open-loop model estimation methods to closed-loop data, not all open-loop model estimation methods handle the correlation between the stimulus signal and output noise well. This chapter describes closed-loop model estimation methods, the corresponding assumptions, and the advantages and disadvantages of these methods.

Feedback in a Closed-Loop System

Feedback is common in control systems. With feedback, the system output corresponds to a reference input. Feedback also reduces the effect of input disturbances. One example of a closed-loop system is a system that regulates room temperature, as shown in Figure 6-1. In this example, the reference input is the temperature T_{set} at which you want the room to stay. The thermostat senses the actual temperature, T_{actual}, of the room. Based on the difference between T_{actual} and T_{set}, the thermostat activates the heater or the air conditioner. The thermostat returns T_{actual} as the feedback to compare again with T_{set}. Then the thermostat uses the difference between T_{actual} and T_{set} to regulate the temperature at the next moment.

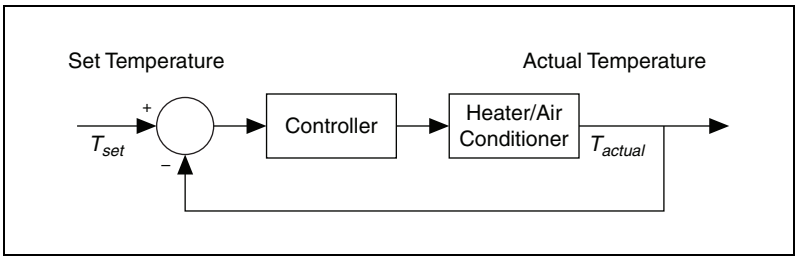

Figure 6-1. A Closed-Loop System Regulating Room Temperature

You must verify if feedback exists before choosing a model estimation method because not all open-loop model estimation methods work correctly with closed-loop data.

Note You need to have knowledge about whether the data you collect is from an open-loop system or a closed-loop system according to the real-world system configuration. If you do not have such information, you can determine if feedback exists by obtaining the impulse response of a plant. You can use the Least Squares instances of the SI Estimate Impulse Response VI to estimate the impulse response of a plant.

Figure 6-2 shows a comparison of the impulse responses of the plant in a closed-loop system and an open-loop system. The values outside the **upper limit** and **lower limit** range at the negative lag, which appears between –10 and 0 on the x-axis, are considered significant values. Significant values in the impulse response at negative lags imply feedback in data. As shown in Figure 6-2, significant values exist in the **Closed-loop data** plot. Therefore, feedback exists in the closed-loop system. No significant impulse response values exist in the **Open-loop data** plot. Thus, feedback does not exist in the open-loop system.

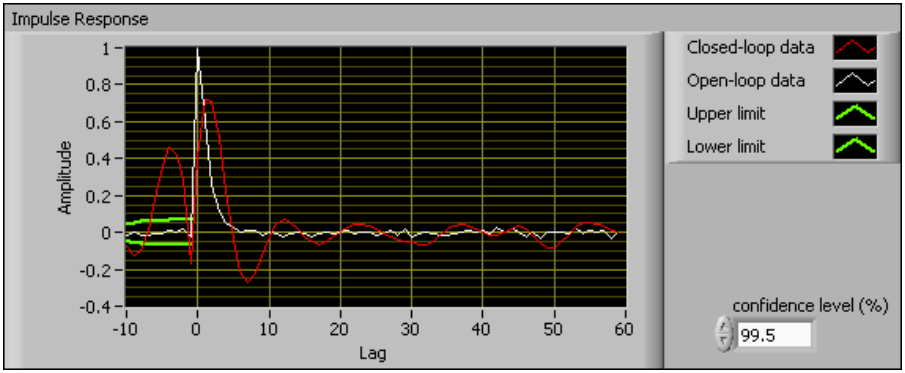

Figure 6-2. Impulse Responses of Open-Loop and Closed-Loop Data

You can use the SI Detect Feedback VI to detect feedback in a system. This VI calculates the impulse response of a plant using the stimulus and response signals of the plant and then calculates upper and lower limits according to the confidence level you set. The SI Detect Feedback VI displays the result of feedback detection in an impulse response graph. If significant values exist in the impulse response at negative lags, the system has feedback. This VI also uses a Boolean indicator to show the result of detection.

Refer to the *Impulse Response* section of Chapter 3, *Nonparametric Model Estimation Methods*, for more information about impulse responses. Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for more information about the SI Estimate Impulse Response VI and the SI Detect Feedback VI.

Understanding Closed-Loop Model Estimation Methods

Closed-loop model estimation methods use data from a closed-loop system to build a model for a plant that a controller regulates. Figure 6-3 shows a system that consists of a plant and a controller. In this system, G_0 is the plant, F_y is the controller, H is the stochastic part of the plant, u is the stimulus signal, y is the response signal, r is the reference signal that is an external signal, and e is the output noise. In control engineering, this system is known as a feedback-path closed-loop system, which is a typical closed-loop system.

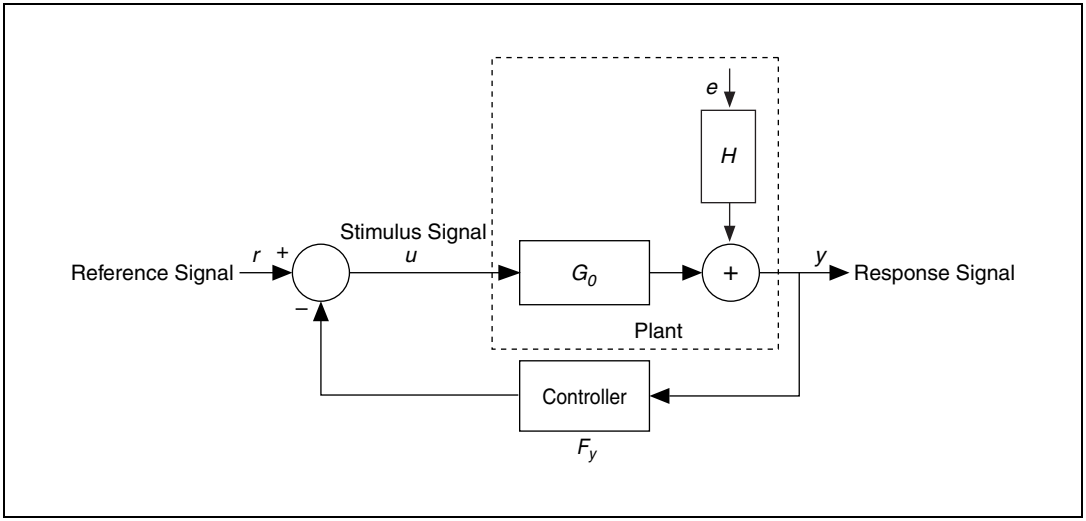

Figure 6-3. A Feedback-Path Closed-Loop System

In some cases, the controller comes before the plant in a closed-loop system. This system is known as a feedforward-path closed-loop system, as shown in Figure 6-4.

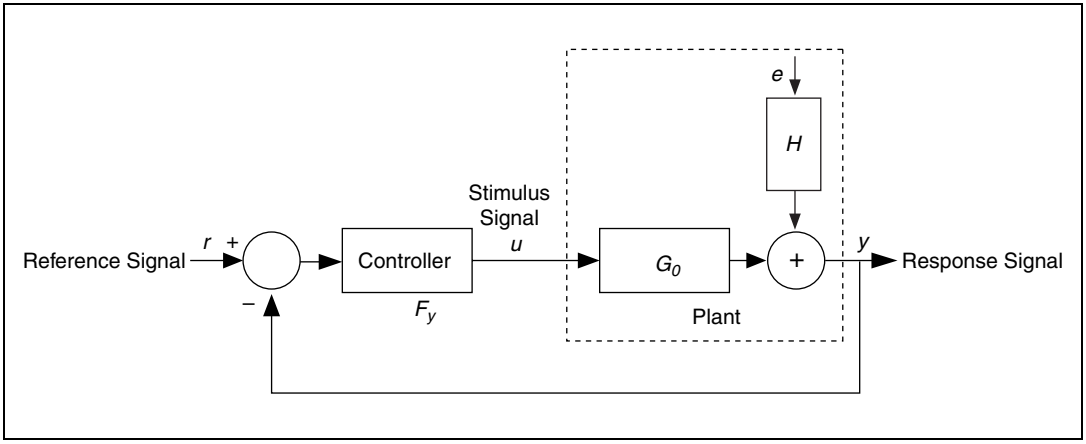

Figure 6-4. A Feedforward-Path Closed-Loop System

Depending on the amount of prior knowledge you have about the feedback, the controller, and the reference signal of a system, you can categorize closed-loop model estimation approaches into the following three groups:

- **Direct identification**—Uses the stimulus signal and the response signal to identify the plant model as if the plant is in an open-loop system. You can apply the direct identification approach to compute many models, such as the general-linear polynomial, state-space, transfer function, and zero-pole-gain models.
- **Indirect identification**—Identifies a closed-loop system using the reference signal and the response signal, and then determines the plant model based on the known controller of the closed-loop system. You can apply the indirect identification approach to compute transfer function models.
- **Joint input-output identification**—Considers the stimulus signal and the response signal as outputs of a cascaded system. The reference signal and the noise jointly perturb the system, and the plant model is identified from this joint input-output system. You can apply the joint input-output identification approach to compute transfer function models.

You can choose a suitable model identification approach according to the information you have about the closed-loop system. Table 6-1 summarizes the information you must have to use each identification approach.

Table 6-1. Required Information for Closed-Loop Identification Approaches

	Stimulus Signal	Response Signal	Reference Signal	Controller Information
Direct	✓	✓	—	—
Indirect	—	✓	✓	✓
Joint Input-Output	✓	✓	✓	—

With the LabVIEW System Identification Toolkit, you can choose to use the direct, indirect, or joint input-output identification approaches for different types of closed-loop systems. The direct identification approach supports single-input single-output (SISO), multiple-input single-output (MISO), and multiple-input multiple-output (MIMO) systems. The indirect and joint input-output identification approaches support SISO systems only.

The following sections discuss the algorithms and assumptions of each closed-loop identification approach in detail.

Direct Identification

If the stimulus and response signals of a closed-loop system are available but you do not have any other information about the system, you can use only the techniques developed for open-loop models to estimate the closed-loop system. However, you cannot apply all open-loop identification methods to estimate the model of a plant in a closed-loop system. Some open-loop model identification methods assume zero correlation between the stimulus signal and output noise. In closed-loop systems, this correlation is nonzero. Thus, if you use certain open-loop model estimation methods, such as the instrument variable (IV) method and the correlation analysis methods, with closed-loop data, you might estimate a model incorrectly. You can use the prediction error method to identify the plant in a closed-loop system.

The direct identification approach is used commonly in real-world applications. This approach is convenient because you do not need to have additional information about a closed-loop system, such as the reference signal or the controller. However, the estimation might not be accurate if the model type you select for a plant does not describe the output noise of the system accurately. For example, if the output noise of a plant is color noise and you select an OE model, which assumes the output noise is white

noise, the estimation for the OE model might be biased when you use direct identification. The bias might be small, though, if the signal-to-noise ratio (SNR) of the system is high.

Indirect Identification

The indirect identification approach, which estimates the transfer function model of a plant in a closed-loop system, first identifies the transfer function model of the closed-loop system based on the reference signal and the response signal. This approach then retrieves the transfer function model of the plant from the identified closed-loop system. The indirect identification approach can identify the transfer function of the plant accurately even when the SNR of the system is low and no matter whether the output noise is white noise or color noise. However, this approach requires prior knowledge about the controller of the system and the reference signal also must be available. In addition, any inaccuracy or nonlinearity of the controller in the system might affect estimating the model of the plant.

With indirect identification, you can use the following two equations to describe the feedback-path closed-loop system shown in Figure 6-3.

$$y(k) = G_0(z)u(k) + e(k) \tag{6-1}$$

$$u(k) = r(k) - F_y(z)y(k) \tag{6-2}$$

where $G_0(z)$ is the open-loop transfer function of the plant

$F_y(z)$ is the transfer function of a linear, time-invariant (LTI) controller

$u(k)$ is the stimulus signal of the system

$y(k)$ is the response signal of the system

$r(k)$ is the reference signal of the system

$e(k)$ is the output noise of the system

By combining Equations 6-1 and 6-2, you can represent the closed-loop relationship with the following equation:

$$y(k) = \frac{G_0(z)}{1 + G_0(z)F_y(z)}r(k) + \frac{1}{1 + G_0(z)F_y(z)}e(k)$$

If you define G_{cl} as the closed-loop transfer function between the reference signal and the response signal, and let G_{cl} satisfy the following equation:

$$G_{cl} = \frac{G_0(z)}{1 + G_0(z)F_y(z)}$$

you can estimate G_{cl} with $r(k)$ as the input and $y(k)$ as the output using an open loop method, because $r(k)$ and $e(k)$ are uncorrelated. You then can calculate G_0 after you calculate G_{cl}, as the following equation shows:

$$G_0 = \frac{G_{cl}}{1 - G_{cl}F_y} \tag{6-3}$$

For feedforward-path closed-loop systems, as shown in Figure 6-4, you use the following two equations to describe the systems.

$$y(k) = G_0(z)u(k) + e(k) \tag{6-4}$$

$$u(k) = [r(k) - y(k)]F_y(z) \tag{6-5}$$

By combining Equations 6-4 and 6-5, you can represent the feedforward-path closed-loop relationship with the following equation:

$$y(k) = \frac{F_y(z)G_0(z)}{1 + G_0(z)F_y(z)}r(k) + \frac{1}{1 + G_0(z)F_y(z)}e(k)$$

If you define G_{cl} as the feedforward-path closed-loop transfer function and let G_{cl} satisfy the following equation:

$$G_{cl} = \frac{F_y(z)G_0(z)}{1 + G_0(z)F_y(z)}$$

you can estimate G_{cl} with $r(k)$ as the input and $y(k)$ as the output using an open loop method, because $r(k)$ and $e(k)$ are uncorrelated. You then can calculate G_0 after you calculate G_{cl}, as the following equation shows:

$$G_0 = \frac{G_{cl}}{(1 - G_{cl})F_y} \tag{6-6}$$

With indirect identification, you calculate G_{cl} by performing polynomial operations on G_o and F_y. Because of the limitations of polynomial operations, the orders of the numerator and denominator might change after

manipulation. Thus, the SI Estimate Transfer Function Model VI or the SI Transfer Function Estimation Express VI, which you can use with the indirect identification approach, might return an error regarding the mismatch between the order you set and the order of the estimated model. In this case, you must adjust the tolerance setting of these two VIs so that the numerator and denominator orders match the orders you set. A larger tolerance facilitates zero-pole cancellations, which reduce the numerator and denominator polynomial orders.

Joint Input-Output Identification

If you do not have any knowledge about the controller structure but the stimulus, response, and reference signals are all available, you can use the joint input-output identification approach to estimate the transfer function model of a plant in a closed-loop system. This approach uses the transfer functions from different input-output signal pairs to estimate a closed-loop system. The System Identification Toolkit implements the following two-stage method for the joint input-output approach.

1. Let $T_0(z)$ satisfy the following equation:

$$T_0(z) = \frac{1}{1 + G_0(z)F_y(z)}$$

By manipulating Equations 6-1 and 6-2, you can rewrite $u(k)$ as follows:

$$u(k) = T_0(z)r(k) - F_y(z)T_0(z)e(k)$$

Any open-loop model estimation method then can estimate $T_0(z)$ because $r(k)$ and $e(k)$ are uncorrelated signals. After you obtain the value of $T_0(z)$, you can compute $\hat{u}(k) = T_0(z)r(k)$. You then can represent $u(k)$ as follows:

$$u(k) = \hat{u}(k) - F_y(z)T_0(z)e(k) \tag{6-7}$$

Using Equation 6-7, you obtain an input signal $\hat{u}(k)$, which is constructed from $r(k)$ and is uncorrelated with the measurement noise.

2. By manipulating Equation 6-1, you can rewrite $y(k)$ as follows:

$$y(k) = G_0(z)\hat{u}(k) + T_0(z)e(k)$$

Because $\hat{u}(k)$ is uncorrelated with $e(k)$, the original closed-loop model estimation problem between $u(k)$ and $y(k)$ becomes an open-loop problem between $\hat{u}(k)$ and $y(k)$.

You use the same methodology to compute $y(k)$ for a feedforward-path closed-loop system, where

$$T_0(z) = \frac{F_y(z)}{1 + G_0(z)F_y(z)}$$

You rewrite $y(k)$ as follows:

$$y(k) = \hat{u}(k)G_0(z) + [1 - T_0(z)G_0(z)]e(k)$$

The two-stage method does not require you to know anything about the feedback or the controller structure and controller parameters. Also, you treat the closed-loop model estimation as an open-loop model estimation within each of the two steps. Therefore, you can use any method that works with open-loop models. Whether the real-world output noise is white noise or color noise, the two-stage method provides reliable estimations.

Using System Identification VIs for Model Estimation

If you want to use the direct identification approach, the System Identification Toolkit enables you to estimate the plant in a closed-loop system with general-linear polynomial, state-space, transfer function, and zero-pole-gain models. If you want to apply the indirect or joint input-output approach to identify the plant, this toolkit provides you with transfer function models. Select the System Identification VIs using the following guidelines:

- Use the Polynomial Model Estimation VIs or the SI Model Estimation Express VI to estimate ARX, ARMAX, output-error, Box-Jenkins, and general-linear models. For ARX models, the System Identification Toolkit uses the least squares method, which is a special case of the prediction error method. For all other models, this toolkit uses the prediction error method. This method can accurately identify a plant model in a closed-loop system. Hence, you can use the Polynomial Model Estimation VIs to estimate the model of a plant in a closed-loop system.
- Use the SI Estimate State-Space Model VI or the SI Model Estimation Express VI to estimate a state-space model of the plant in a closed-loop system. You can choose the deterministic-stochastic subspace method that the SI Estimate State-Space Model VI implements. The deterministic-stochastic subspace method uses principal component analysis to estimate parameters. This method does not assume a zero correlation between the input signal and the output noise.

Therefore, the deterministic-stochastic subspace method works with data from a closed-loop system.

- Use the SI Estimate Transfer Function VI or the SI Transfer Function Estimation Express VI to estimate a transfer function model of the plant in a closed-loop system. You can apply direct, indirect, and joint input-output identification to compute transfer function models.
- To identify zero-pole-gain models for a plant, you first must identify the plant using other model representations. You then can convert other model representations to zero-pole-gain models using the Model Conversion VIs. Refer to Chapter 8, *Analyzing, Validating, and Converting Models*, for more information about converting models.

Refer to Chapter 4, *Parametric Model Estimation Methods*, for more information about the models and model algorithms described in this section. Refer to the *LabVIEW System Identification Toolkit Algorithm References* manual (`SIreference.pdf`), available in the `labview\manuals` directory, for more information about the prediction error method, the deterministic-stochastic subspace method, and the realization method. Refer to the *LabVIEW Help* for more information about the VIs described in this section.

7

Recursive Model Estimation Methods

The model estimation methods in Chapter 3, *Nonparametric Model Estimation Methods*, Chapter 4, *Parametric Model Estimation Methods*, Chapter 5, *Partially Known Model Estimation Methods*, and Chapter 6, *Model Estimation Methods in Closed-Loop Systems*, use nonrecursive methods to estimate a model of the plant in a system. Nonrecursive model estimation identifies a model for a plant based on input-output data gathered at a time prior to the current time. However, many real-world applications such as adaptive control and adaptive prediction, having a model of the system update while the system is running is necessary or helpful. In this type of application, you obtain the mathematical model of the system in real time.

Recursive model estimation is a common system identification technique that enables you to develop a model that adjusts based on real-time data coming from the system. Recursive model estimation processes the measured input-output data recursively as the data becomes available. This chapter discusses recursive model estimation techniques and various adaptive algorithms associated with each method.

Defining Recursive Model Estimation

Figure 7-1 represents a general recursive system identification application. A system identification application consists of an unknown system that has an input signal, or stimulus signal $u(k)$ and an output signal, or response signal $y(k)$.

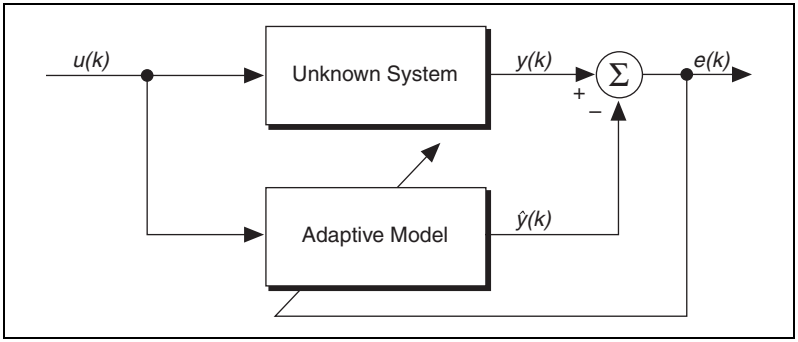

Figure 7-1. Recursive System Identification Diagram

The stimulus signal $u(k)$ is the input to both the unknown system and the recursive model. The response of the system $y(k)$ and the predicted response of the adaptive model $\hat{y}(k)$ are combined to determine the error of the system. The error of the system is defined by the following equation.

$$e(k) = y(k) - \hat{y}(k) \tag{7-1}$$

The adaptive model generates the predicted response $\hat{y}(k+1)$ based on $u(k+1)$ after adjusting the parametric vector $\vec{w}(k)$ based on the error $e(k)$. Refer to the *Adaptive Algorithms* section of this chapter for more information about the definition of $\vec{w}(k)$ in different algorithms.

Figure 7-1 shows how the error information $e(k)$ is sent back to the adaptive model, which adjusts the parametric vector $\vec{w}(k)$ to account for the error. You iterate on this process until you minimize the magnitude of the least mean square error $e(k)$.

Before you apply the recursive model estimation, you must first select the parametric model structure that determines the parametric vector $\vec{w}(k)$. Then, you must select the method that automatically adjusts the parametric vector such that the error $e(k)$ goes to the minimum.

The LabVIEW System Identification Toolkit provides Recursive Model Estimation VIs that support the following model structures:

- ARX
- ARMAX
- Output-Error
- Box-Jenkins
- General-Linear

The Recursive Model Estimation VIs have a **recursive method** parameter that enables you to specify which recursive estimation method to use.

Refer to the *Parametric Model Representations* section of Chapter 4, *Parametric Model Estimation Methods*, for information about each of these models. Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for more information about the Recursive Model Estimation VIs and mathematical definitions of each of these models.

You can compute each model recursively using the following four types of adaptive algorithms:

- Least mean square (LMS)
- Normalized least mean squares (NLMS)
- Recursive least squares (RLS)
- Kalman filter (KF)

The following sections provide more information about each of these adaptive algorithms.

Adaptive Algorithms

Adaptive algorithms are fundamental in recursive system identification. The adaptive method you use affects the performance of recursive system identification application.

The goal of all recursive algorithms is to adjust the parametric vector $\vec{w}(k)$ until you minimize the cost function $J(k)$. The following equation defines the cost function $J(k)$.

$$J(k) = E[e^2(k)]$$

where E is the expectation.

Again, $e(k)$ represents the difference between the predicted response $\hat{y}(k)$ and the response $y(k)$ of the unknown system, as shown in Figure 7-1.

When the cost function $J(k)$ is sufficiently small, the parametric vector $\vec{w}(k)$ is considered optimal for the estimation of the actual system.

Least Mean Squares

The least mean squares (LMS) method uses the following equations to define the cost function $J(k) = E[e^2(k)]$.

The parametric vector $\vec{w}(k)$ updates according to the following equation.

$$\vec{w}(k+1) = \vec{w}(k) + \mu e(k)\vec{\varphi}(k)$$

k is the number of iterations, μ is step-size, which is a positive constant, and $\vec{\varphi}(k)$ is the data vector from the past input data $u(k)$ and output data $y(k)$. $\vec{\varphi}(k)$ is defined by the following equation.

$$\varphi(k) = \left[-y(t-1)\ldots -y(t-k)\ u(t-1)\ldots u(t-m)\right]^T$$

The following procedure describes how to implement the LMS algorithm.

1. Initialize the step-size μ.
2. Initialize the parametric vector $\vec{w}(k)$ using a small positive number ε.

$$\vec{w}(0) = [\varepsilon, \varepsilon, \ldots, \varepsilon]^T$$

3. Initialize the data vector $\vec{\varphi}(k) = \left[u(k)\ y(k)\right]$.

$$\vec{\varphi}(0) = [0, 0, \ldots, 0]^T$$

4. For $k = 1$, update the data vector $\vec{\varphi}(k)$ based on $\vec{\varphi}(k-1)$ and the current input data $u(k)$ and output data $y(k)$.
5. Compute the predicted response $\hat{y}(k)$ using the following equation.

$$\hat{y}(k) = \vec{\varphi}^T(k)\vec{w}(k)$$

6. Compute the error $e(k)$ by solving the following equation.

$$e(k) = y(k) - \hat{y}(k)$$

7. Update the parameter vector $\vec{w}(k)$.

$$\vec{w}(k+1) = \vec{w}(k) + \mu e(k)\vec{\varphi}(k)$$

8. Stop if the error is small enough, else set $k = k + 1$ and repeat steps 4–8.

The LMS algorithm is one of the most widely used and understood adaptive algorithms. Selecting the step-size μ is important with the LMS algorithm, because the selection of the step-size μ directly affects the rate of convergence and the stability of the algorithm. The convergence rate of the LMS algorithm is usually proportional to the step-size μ. The larger the step-size μ, the faster the convergence rate. However, a large step-size μ can cause the LMS algorithm to become unstable. The following equation describes the range of the step-size μ.

$$0 < \mu < \mu_{max}$$

μ_{max} is the maximum step-size that maintains stability in the LMS algorithm. μ_{max} is related to the statistical property of the stimulus signal. A uniformly optimized step-size μ that achieves a fast convergence speed while maintaining the stability in the system does not exist, regardless of the statistical property of the stimulus signal. For better performance, use a self-adjustable step-size μ and the normalized least mean squares (NLMS) algorithm.

Normalized Least Mean Squares

The following equation defines a popular self-adjustable step-size $\mu(k)$ that you use in the normalized least mean squares algorithm.

$$\mu(k) = \frac{\mu}{\varepsilon + \left\| \vec{\varphi}(k) \right\|^2}$$

Again, $\vec{\varphi}(k)$ represents the data vector. ε is a very small positive number that prevents the denominator from equaling zero when $\left\| \vec{\varphi}(k) \right\|^2$ approaches zero.

The step-size $\mu(k)$ is time-varying because the step-size changes with the time index k.

Substituting $\mu(k)$ into the parametric vector $\vec{w}(k)$ equation yields the following equation.

$$\vec{w}(k+1) = \vec{w}(k) + \mu(k)e(k)\vec{\varphi}(k)$$

Compared to the LMS algorithm, the NLMS algorithm is always stable if the step-size $\mu(k)$ is between zero and two, regardless of the statistical property of the stimulus signal $u(k)$.

The procedure of the NLMS algorithm is the same as the LMS algorithm except for the estimation of the time-varying step-size $\mu(k)$.

Recursive Least Squares

The recursive least squares (RLS) algorithm and Kalman filter algorithm use the following equations to modify the cost function $J(k)$ defined in the *Adaptive Algorithms* section. Refer to the *Kalman Filter* section for information about the Kalman filter algorithm.

$$J(k) = E[e^2(k)] \cong \frac{1}{N}\sum_{i=0}^{N-1} e^2(k-i)$$

Compare this modified cost function, which uses the previous N error terms, to the cost function, $J(k) = E[e^2(k)]$, which uses only the current error information $e(k)$. The modified cost function $J(k)$ is more robust. The corresponding convergence rate in the RLS algorithm is faster, but the implementation is more complex than that of LMS-based algorithms.

The following procedure describes how to implement the RLS algorithm.

1. Initialize the parametric vector $\vec{w}(k)$ using a small positive number ε.

$$\vec{w}(0) = [\varepsilon, \varepsilon, \ldots, \varepsilon]^T$$

2. Initialize the data vector $\vec{\varphi}(k)$.

$$\vec{\varphi}(0) = [0, 0, \ldots, 0]^T$$

3. Initialize the $k \times k$ matrix $\boldsymbol{P}(0)$.

$$\boldsymbol{P}(0) = \begin{bmatrix} \varepsilon & 0 & 0 & 0 \\ 0 & \varepsilon & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \varepsilon \end{bmatrix}$$

4. For $k = 1$, update the data vector $\vec{\varphi}(k)$ based on $\vec{\varphi}(k-1)$ and the current input data $u(k)$ and output data $y(k)$.
5. Compute the predicted response $\hat{y}(k)$ by using the following equation.

$$\hat{y}(k) = \vec{\varphi}^T(k) \cdot \vec{w}(k)$$

6. Compute the error $e(k)$ by solving the following equation.

$$e(k) = y(k) - \hat{y}(k)$$

7. Update the gain vector $\vec{K}(k)$ defined by the following equation.

$$\vec{K}(k) = \frac{\boldsymbol{P}(k) \cdot \vec{\varphi}(k)}{\lambda + \vec{\varphi}^T(k) \cdot \boldsymbol{P}(k) \cdot \vec{\varphi}(k)}$$

The properties of a system might vary with time, so you need to ensure that the algorithm tracks the variations. You can use the forgetting factor λ, which is an adjustable parameter, to track these variations. The smaller the forgetting factor λ, the less previous information this algorithm uses. When you use small forgetting factors, the adaptive filter is able to track time-varying systems that vary rapidly. The range of the forgetting factor λ is between zero and one, typically $0.98 < \lambda < 1$.

$\boldsymbol{P}(k)$ is a $k \times k$ matrix whose initial value is defined by $\boldsymbol{P}(0)$ in step 3.

8. Update the parametric vector $\vec{w}(k+1)$.

$$\vec{w}(k+1) = \vec{w}(k) + e(k) \cdot \vec{K}(k)$$

9. Update the $\boldsymbol{P}(k)$ matrix.

$$\boldsymbol{P}(k+1) = \boldsymbol{P}(k) - \vec{K}(k) \cdot \vec{\varphi}^T(k) \cdot \boldsymbol{P}(k)$$

10. Stop if the error is small enough, else set $k = k + 1$ and repeat steps 4–10.

Kalman Filter

The Kalman filter is a linear optimum filter that minimizes the mean of the squared error recursively. The convergence rate of the Kalman filter is relatively fast, but the implementation is more complex than that of LMS-based algorithms.

Recall that the equation $J(k) = E[e^2(k)]$ defines the cost function. The following procedure lists the steps of the Kalman filter algorithm.

1. Initialize the parametric vector $\vec{w}(k)$ using a small positive number ε.

$$\vec{w}(0) = [\varepsilon, \varepsilon, \ldots, \varepsilon]^T$$

2. Initialize the data vector $\vec{\varphi}(k)$.

$$\vec{\varphi}(0) = [0, 0, \ldots, 0]^T$$

3. Initialize the $k \times k$ matrix $\boldsymbol{P}(0)$.

$$\boldsymbol{P}(0) = \begin{bmatrix} \varepsilon & 0 & 0 & 0 \\ 0 & \varepsilon & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \varepsilon \end{bmatrix}$$

4. For $k = 1$, update the data vector $\vec{\varphi}(k)$ based on $\vec{\varphi}(k-1)$ and the current input data $u(k)$ and output data $y(k)$.
5. Compute the predicted response $\hat{y}(k)$ by solving the following equation.

$$\hat{y}(k) = \vec{\varphi}^T(k) \cdot \vec{w}(k)$$

6. Compute the error $e(k)$ by solving the following equation.

$$e(k) = y(k) - \hat{y}(k)$$

7. Update the Kalman gain vector $\vec{K}(k)$ defined by the following equation.

$$\vec{K}(k) = \frac{\boldsymbol{P}(k) \cdot \vec{\varphi}(k)}{\boldsymbol{Q}_M + \vec{\varphi}^T(k) \cdot \boldsymbol{P}(k) \cdot \vec{\varphi}(k)}$$

$\boldsymbol{Q}_M$ is the measurement noise and $\boldsymbol{P}(k)$ is a $k \times k$ matrix whose initial value is defined by $\boldsymbol{P}(0)$ in step 3.

8. Update the parametric vector $\vec{w}(k)$.

$$\vec{w}(k+1) = \vec{w}(k) + e(k) \cdot \vec{K}(k)$$

9. Update the $\boldsymbol{P}(k)$ matrix.

$$\boldsymbol{P}(k+1) = \boldsymbol{P}(k) - \vec{K}(k) \cdot \vec{\varphi}^T(k) \cdot \boldsymbol{P}(k) + \boldsymbol{Q}_P$$

$\boldsymbol{Q}_P$ is the correlation matrix of the process noise.

10. Stop if the error is small enough, else set $k = k + 1$ and repeat steps 4–10.

8

Analyzing, Validating, and Converting Models

After estimating a model for the plant of a system, you can observe model characteristics by analyzing the model. You also can verify that the model simulates the real-world plant by validating the model. The LabVIEW System Identification Toolkit provides tools that enable you to analyze and validate models.

According to linear system theory, you can represent a linear system with different models. Each model representation has benefits and drawbacks for characterizing a dynamic system. Certain model representations are more suitable for certain analysis techniques. With the System Identification Toolkit, you have the flexibility to convert models from one representation to another to identify the best-fit model for the system.

This chapter describes model analysis and validation methods. This chapter also describes how to convert models with the Model Conversion VIs.

Analyzing Models

Model analysis allows you to observe some characteristics, like frequency response, stability, and order, of the model you obtain. The System Identification Toolkit enables you to investigate model estimation results and present these results in graphs with the following three tools—the Bode plot, the Nyquist plot, and the pole-zero plot.

Bode Plot

The Bode plot contains a Bode magnitude plot and a Bode phase plot. The Bode magnitude plot describes magnitude against frequency and the Bode phase plot describes phase against frequency. These two plots together describe the frequency response of the plant model you estimate, as shown in Figure 8-1.

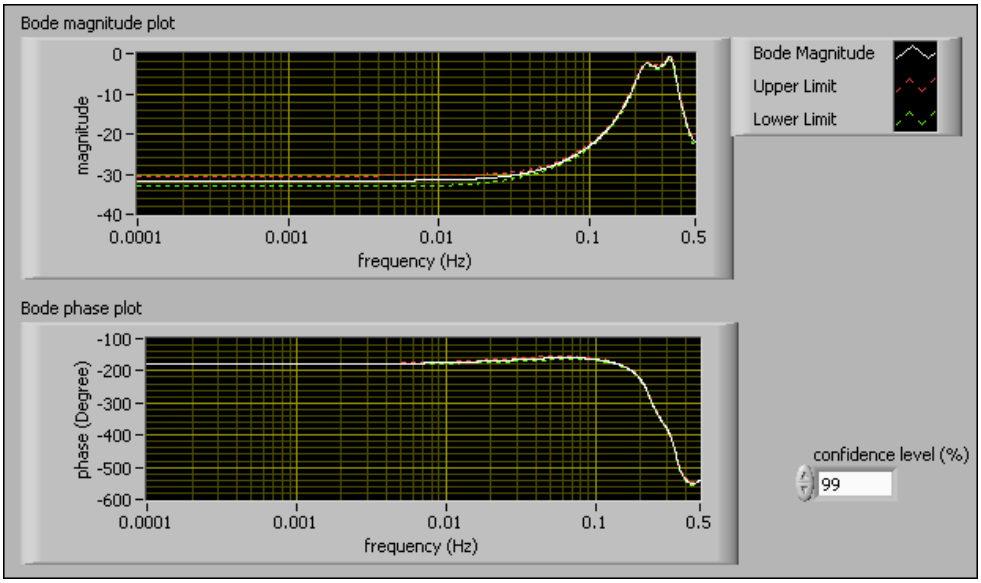

Figure 8-1. A Bode Plot with Sample Data

The SI Bode Plot VI also calculates the upper and lower limits according to the confidence level you set. You can obtain information, such as the gain of the system and the cutoff frequency, by evaluating the Bode plot. You can use the SI Bode Plot VI to produce the Bode magnitude and Bode phase plots. You then can display the Bode magnitude and phase using the SI Bode Plot Indicator.

Nyquist Plot

The Nyquist plot describes the gain and phase of a frequency response in polar coordinates by plotting the imaginary part of the complex frequency response against the real part. You can view the Nyquist plot as a combination of the Bode magnitude plot and the Bode phase plot. In polar coordinates, the Nyquist plot shows the phase as the angle and the magnitude as the distance from the origin, as shown in Figure 8-2.

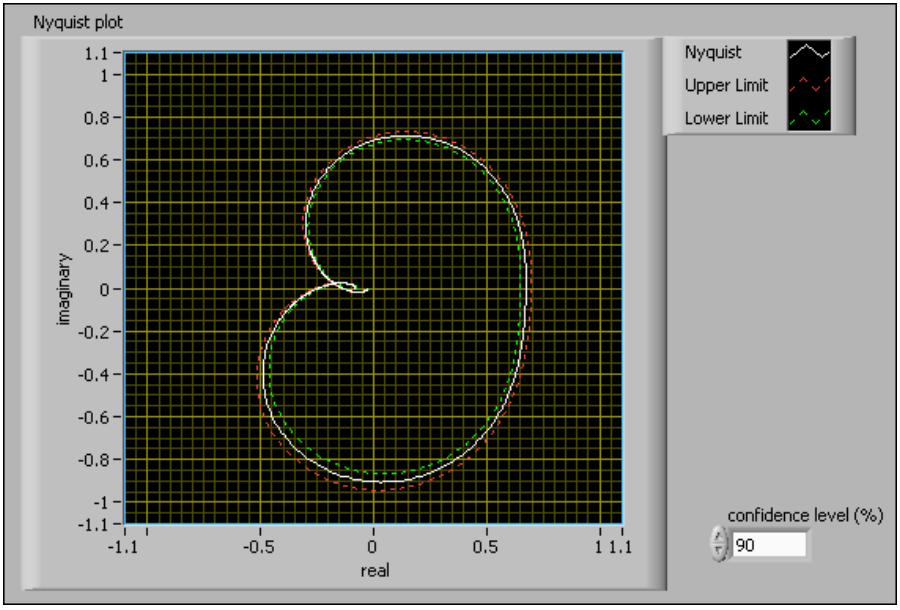

Figure 8-2. A Nyquist Plot with Sample Data

The SI Nyquist Plot VI also calculates the upper and lower limits according to the confidence level you set. You can use the SI Nyquist Plot VI to generate the Nyquist plot and display this plot using the SI Nyquist Plot Indicator. The Nyquist plot is commonly used to predict the stability of a system.

Pole-Zero Plot

The pole-zero plot displays the poles and zeros of a system. By observing the locations of the poles and zeros, you can conclude if the system is stable. In a stable system, all the poles fall within the unit circle. Figure 8-3 shows an example of a stable model.

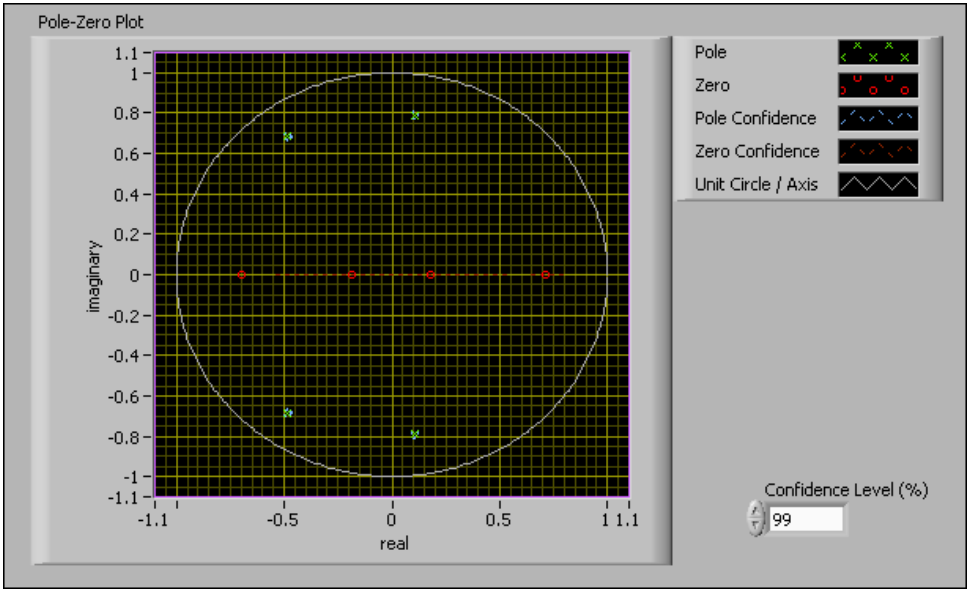

Figure 8-3. A Pole-Zero Plot with Sample Data

You can use the SI Pole-Zero Plot VI to generate the pole-zero plot and display this plot using the SI Pole-Zero Plot Indicator.

You also can use the pole-zero plot to determine if you can reduce model orders. By observing the pole-zero placements, you can determine if any pole-zero pairs have overlapping confidence intervals. A confidence interval is a region the SI Pole-Zero Plot VI calculates from the confidence level you set. The existence of overlapping confidence intervals implies that pole-zero cancellations exist and that the model order might be unnecessarily high. The pole-zero plot shown in Figure 8-3 is an optimal model with the appropriate order because the pole-zero pairs do not have overlapping confidence intervals.

If the model order is too high, you can try reducing the model order. You then can use the F-test criterion to assess if the reduction in model order leads to a significant increase in the prediction error. If the reduction in

model order leads to a significant increase in the prediction error, do not reduce the model order.

Validating Models

Model estimation determines the best model of the system within the chosen model structure. Model estimation does not determine if the model provides the most accurate description of the system. After you obtain a model, you must validate the model to determine how well the behavior of the model corresponds to the data you measured, to any prior knowledge of the system, and to the purpose for which you use the model. Model validation also determines if the model is flexible enough to describe the system. If the model is inadequate, you must revise the system identification process or consider using another method.

The best way to validate a model is to experiment with the model under real-world conditions. If the model works as you expect, the model estimation is successful. However, experimenting with the model under real-world conditions might be dangerous. For example, introducing arbitrary perturbations to the input of a chemical plant might lead to a harmful explosion. Therefore, before you incorporate the model into real-world applications, validate the model by using plots and common sense or by using statistical tests on the prediction error.

The System Identification Toolkit provides three of the most common validation methods—model simulation, model prediction, and model residual analysis.

Note When validating the model you obtain, you must use a set of data that is different from the data you used to estimate the model.

Validation Methods

After you build a model, you can use at least three different methods to validate the model and evaluate its flexibility. You can use model simulation to understand the underlying dynamic relationship between the model inputs and outputs. You can use model prediction to test the ability of the model to predict the response of the system using past input and output data. You also can use model residual analysis to test, using statistical techniques, the whiteness of the prediction error and the independency between the prediction error and the input signal. The methods you select to validate the model depend on the purpose for which you created the model. You can use one or all of these methods to validate the model.

Model Simulation

The SI Model Simulation VI determines the outputs of a system for given inputs. After you build a model for the system using the input and output data you measured, you can use the model to simulate the response of the system by using the model equations. You then can evaluate the behavior of the system. You also can use simulation to validate the model by comparing the simulated response with the measured response.

Model Prediction

The SI Model Prediction VI determines the response of a system at time t based on the output information available at time $t - k$ and all the inputs applied from time $t - k$ to time t. k represents the size of the prediction window. Therefore, model prediction can determine how useful a model is in estimating future responses of the system, given all information at time t and an expected input profile in the future. Some control techniques take advantage of model prediction to improve control performance. For example, model predictive control uses some of the prediction properties of a model to determine if a particular limitation or constraint is active in the future. This method allows the controller to take preventive actions before such constraints become active.

If you have the measured input and output of a system, you also can validate the model of the system by comparing the predicted output and the measured output. If the prediction error is small, the model is acceptable.

Residual Analysis

Residual analysis is the third validation method that the System Identification Toolkit provides. The response that an estimated model predicts and the actual response from the system are different. This difference is called the prediction error or residual. The following equation defines the residual $e(k)$.

$$e(k) = y(k) - y'(k)$$

$y(k)$ is the measured output and $y'(k)$ is the output from the one-step-ahead prediction. If the model is capable of describing the true system, the residual is zero-mean white noise and independent of the input signal. You can use autocorrelation analysis to test if the residual is zero-mean white noise. You can use cross correlation analysis to test if the residual is independent of the input signal. The SI Model Residual Analysis VI calculates both the autocorrelation and the cross correlation values.

Autocorrelation

The following equation defines the autocorrelation of the residuals.

$$R_e{}^N(\tau) = \frac{1}{N}\sum_{k=1}^{N} e(k)e(k-\tau)$$

Ideally, the residual is white noise, and therefore the autocorrelation $R_e{}^N(\tau)$ is zero when τ is nonzero. A large autocorrelation when τ is nonzero indicates that the residual is not zero-mean white noise and also implies that the model structure is not relevant to the system or that you might need to increase the model order.

In real-world applications, the autocorrelation $R_e{}^N(\tau)$ cannot be zero when τ is nonzero because of the limited length of data points. However, the SI Model Residual Analysis VI assesses if the autocorrelation value is sufficiently small to be ignored. If the value of autocorrelation falls within the confidence range, the autocorrelation value is insignificant and you can consider this value to be equal to zero.

Cross Correlation

The following equation defines the cross correlation between residuals and past inputs.

$$R_{eu}{}^N(\tau) = \frac{1}{N}\sum_{k=1}^{N} e(k)u(k-\tau)$$

If the residual is independent of the input, the cross correlation is zero for all τ. If the residual correlates with the input, the cross correlation is nonzero, suggesting that the model did not capture all deterministic variations from the data. Therefore, you need to revise the model variation.

The SI Model Residual Analysis VI assesses if the value of cross correlation is sufficiently small. If the value of cross correlation falls within the confidence range, the value is insignificant and you can consider this value to be equal to zero.

Converting Models

With the System Identification Toolkit, you can apply the Model Conversion VIs to convert system models from one representation to another, from continuous to discrete models, and from discrete to continuous models. You can convert models you created in this toolkit to models you can use in another toolkit. You also can convert a model after you estimate the model or after you analyze or validate the model.

You can use the Model Conversion VIs to switch between different model representations or types. For example, when estimating a digital system, you can convert an existing continuous model to a discrete model to approximate the real-time behavior of the system. You do not need to create a new discrete model for the digital system. Using the Model Conversion VIs, you also can convert models you create in the System Identification Toolkit into transfer function, zero-pole-gain, or state-space models that you then can use with the LabVIEW Control Design Toolkit. This model conversion process enables you to identify a model for an unknown system with the System Identification Toolkit and then design a controller for this system using the Control Design Toolkit.

Refer to the *Model-Based Control Design Process* section of Chapter 1, *Introduction to System Identification*, for more information about the integration of the System Identification Toolkit and the Control Design Toolkit in the model-based control design process. Refer to the National Instruments Web site at `ni.com` for more information about the Control Design Toolkit. Refer to the *LabVIEW Help*, available by selecting **Help» Search the LabVIEW Help**, for more information about the VIs described in this chapter.

9

System Identification Case Study

This chapter contains a case study that guides you through the system identification process. The case study uses sample data that the LabVIEW System Identification Toolkit provides in the SI Data Samples VI. The SI Data Samples VI includes data sets for a DC motor, a flexible robot[1] arm, a ball and beam apparatus, an RC circuit, and so on. The case study in this chapter uses the flexible arm data to demonstrate the system identification process and to compare different estimation methods.

The flexible arm is a nonlinear dynamic system. The System Identification Toolkit enables you to build models for systems linearly. This chapter guides you through obtaining a linear representation of a nonlinear system.

The VIs for this case study are located in the `labview\examples\system identification\SICaseStudy1.llb`. Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for more information about specific System Identification VIs in this case study.

Data Preprocessing

After you gather data, the next step in the system identification process is to preprocess the data. The input to the system in this case study is the reaction torque of the structure on the ground. This input is a swept sine wave with 200 frequency points equally spaced over the frequency band from 0.122 Hz to 24.4 Hz.

The output of this system is the acceleration of the flexible arm. The acceleration contains information about the flexible resonances and anti-resonances.

[1] The flexible robotic arm data was adopted from a case study in the *MATRIXx Interactive System Identification Module, Part 2* manual. Hendrik Van Brussel and Jan Swevers of the laboratory of Production Manufacturing and Automation of the Katholieke Universiteit Leuven provided this data, which they obtained in the framework of the Belgian Programme on Interuniversity Attracion Poles.

The data set contains 4096 samples at a sampling rate of 500 Hz or sampling time of 0.002 seconds. Thus the total time of the response is 8.192 seconds.

The following sections show you how to preprocess the raw data by examining the time and frequency responses of the system. Based on those analyses, you can filter and downsample the data set to reduce the amount of data in the raw data set for simpler identification.

Examining the Time Response Data

Using the data in the SI Data Sample VI for the flexible robotic arm, you can view the input and output data, as shown in Figure 9-1.

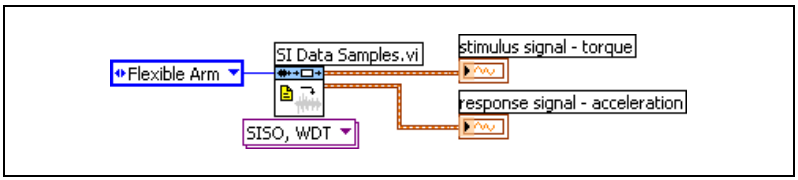

Figure 9-1. Flexible Arm Data Set VI

Note The names of the block diagram figures in this chapter reflect the names of the example VIs located in the `labview\examples\system identification\SICaseStudy1.llb`.

The **stimulus signal – torque** output corresponds to the input data, or the torque, and the **response signal – acceleration** output corresponds to the output data, or the acceleration.

Figure 9-2 shows the input and output data on graphs during the length of the response. By looking at the graphs, you can inspect the data for outliers, clipped saturation, or quantization effects that you can remove because they are not representative of the system behavior.

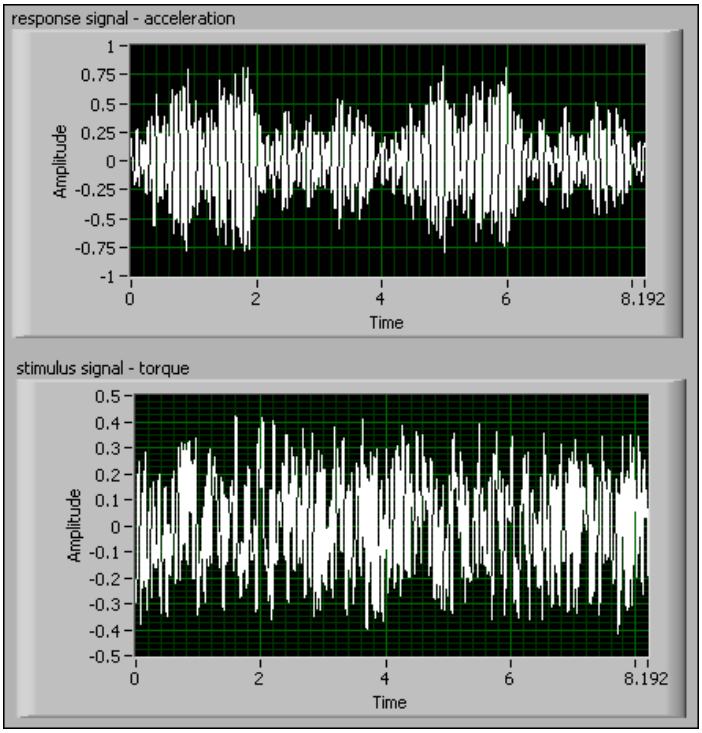

Figure 9-2. Flexible Arm Data Set Plotted in the Time Domain

Figure 9-2 shows no obvious nominal, trend, or outlier values in the input or output time waveforms.

Examining the Frequency Response Data

In addition to examining the time response data, you also want to examine the frequency response data. You can use the SI Estimate Frequency Response VI to view the frequency response of the measured output signal, as shown in Figure 9-3.

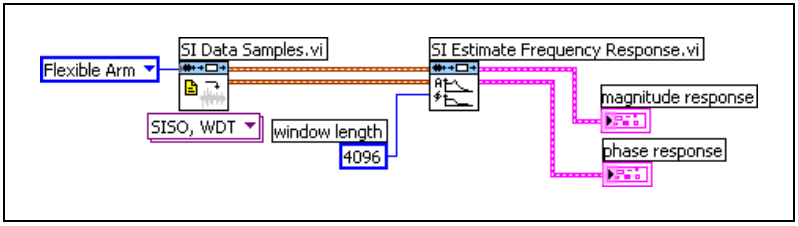

Figure 9-3. Non-Parametric FRF VI

The input data is periodic over 4096 samples, which is the signal length. Notice that in Figure 9-3 the **window length**, 4096, is the same as the signal length so as to obtain a smaller bias in the frequency response estimation.

Figure 9-4 shows the magnitude and phase responses of the measured output signal. The **magnitude response** graph shows three resonances and two anti-resonances in the frequency domain. Resonances are vibrations of large amplitude in a system caused by exciting the system at its natural frequency.

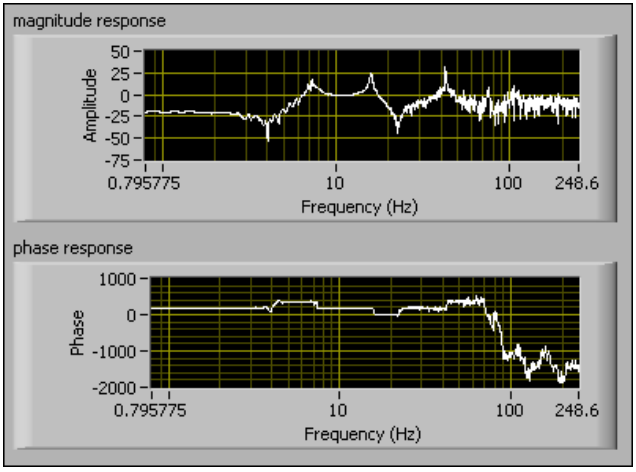

Figure 9-4. Frequency Response of the Flexible Arm Data Set

Notice the resonance at approximately 42 Hz. You can deduce that this resonance is caused by noise or nonlinear system behavior because the 42 Hz falls outside the frequency range of the input data, 0.122–24.4 Hz. At 42 Hz, there is no input energy, thus implying that the response at 42 Hz is not a result of the input.

By examining the frequency response data, you see that filtering is necessary to remove this resonance peak at 42 Hz. The following section describes how to use the System Identification Toolkit to apply a filter to the flexible arm data.

Applying a Filter to the Raw Data

To eliminate the resonance peak at 42 Hz, you can apply a filter to the raw data. By first applying a lowpass filter with a cutoff frequency of 25 Hz, you eliminate the high-frequency noise from the raw data set. Figure 9-5 shows how to use SI Lowpass Filter to apply a lowpass filter to the raw data set.

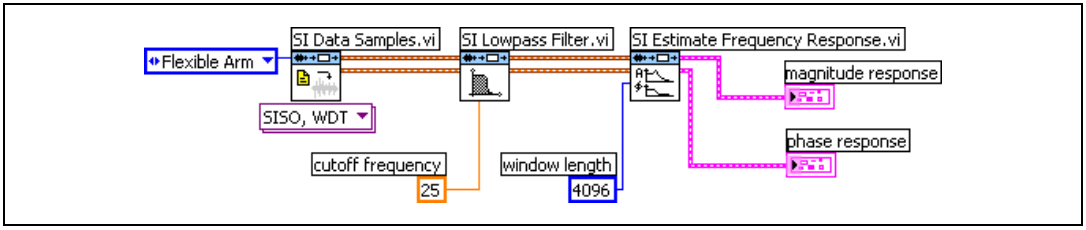

Figure 9-5. Non-Parametric FRF with Prefiltering VI

You can see the effects of the lowpass filter by comparing the frequency response of the filtered data set in Figure 9-6 to the frequency response of the non-filtered data set in Figure 9-4. By using a lowpass filter, you can see that the resonance at approximately 42 Hz is no longer part of the data set you will use to estimate the model.

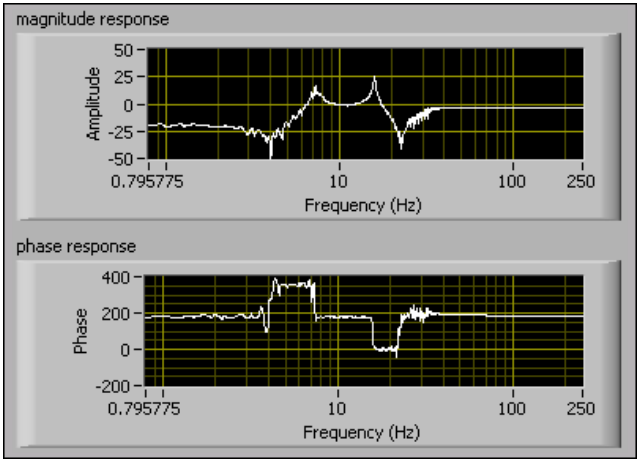

Figure 9-6. Frequency Response of the Filtered Data Set

Downsampling the Raw Data

Sampling theory, in conjunction with the Nyquist criterion, enables you to reduce the sampling rate from 500 Hz to 50 Hz. Applying a filter and downsampling the data set reduces the number of samples in and the computational complexity of the data set. The goal is to use as few samples as possible to evaluate the behavior of the system.

Sampling theory enables you to downsample, or decimate, the data set. Downsampling reduces the sampling rate, 500 Hz, by a factor of 10. Thus downsampling enables you to acquire the data at a sampling rate of 50 Hz. The Nyquist criterion states that you need to sample the signal at a minimum of twice the highest frequency in the system.

Recall that the input data is equally spaced over the frequency band 0.122–24.4 Hz. Therefore, according to the Nyquist criterion, you need to sample at a minimum of 50 Hz to avoid any antialiasing. The benefit of sampling at 50 Hz is that you still acquire all the data in the frequency band, yet you eliminate the resonance peak at 42 Hz.

Therefore, in Figure 9-7, the SI Lowpass Filter VI sets the **cutoff frequency** to 25. In addition to applying a lowpass filter to the data, you must downsample the reduced data set. The SI Down Sample VI in Figure 9-7 uses a **decimation factor** of 10.

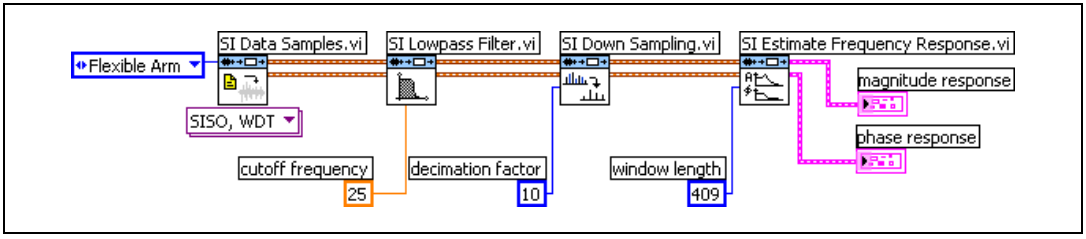

Figure 9-7. Non-Parametric FRF with Prefiltering and Down Sampling VI

The SI Lowpass Filter VI applies a lowpass filter before downsampling the data set to avoid aliasing at the 42 Hz resonance. Together, the lowpass filter and downsampling remove the high frequency disturbance and make the process faster and more efficient.

Notice that the **window length** parameter of the SI Estimate Frequency Response VI in Figure 9-7 is around 400 instead of 4096, as shown in Figure 9-3. You can reduce the window length by a factor of 10 because the number of samples in the reduced data set is one tenth of the number of samples in the raw data set.

Figure 9-8 shows the frequency response after applying a filter to and downsampling the raw data set.

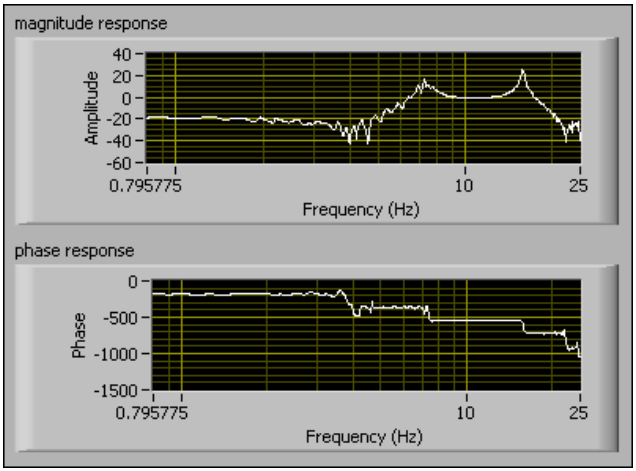

Figure 9-8. Frequency Response of the Filtered Data Set after Downsampling

Filtering and downsampling are beneficial because they eliminate the nonrealistic parts of the frequency response and reduce the amount of work required in the model estimation process.

Refer to Chapter 2, *Acquiring and Preprocessing Data*, for more information about filtering and downsampling data.

Estimating the Model

One of the biggest challenges in model estimation is selecting the correct model and the order of the model. The System Identification Toolkit supports three different criteria to aid in the estimation of the order of a model.

- FPE—Final Prediction Error Criterion
- AIC—Akaike's Information Criterion
- MDL—Minimum Description Length criterion

Sometimes the results you obtain with these three criteria might be inconsistent. You can use a pole-zero plot for further investigation and to verify the results of the order estimation. Refer to the *Akaike's Information Criterion* section, the *Final Prediction Error Criterion* section, and the

Minimum Description Length Criterion section of Chapter 4, *Parametric Model Estimation Methods*, for more information about these criteria.

Figure 9-9 shows a prediction error plot generated by the SI Estimate Orders of System Model VI for an ARX model. The y-axis is the prediction error and the x-axis is the model dimension. The three different color bars on the chart represent the FPE, AIC, and MDL criteria.

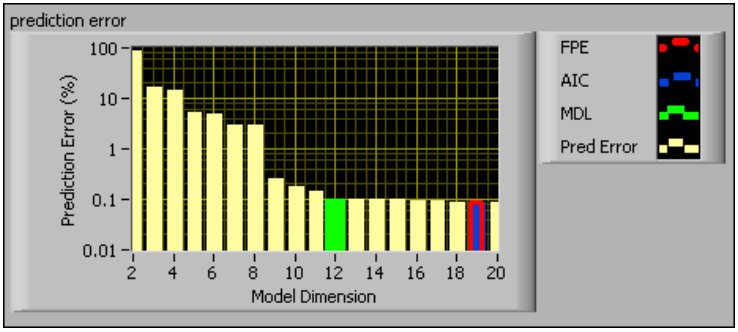

Figure 9-9. Prediction Error Plot for an ARX Model

The following sections show you how to use the AIC, MDL, and a user-defined criterion to determine the A and B orders of the ARX model.

Akaike's Information Criterion

The block diagram in Figure 9-10 uses the SI Estimate Orders of System Model VI for order estimation. To estimate the orders of a model, the SI Estimate Orders of System Model VI requires two data sets—one for estimation and one for validation. You do not need to acquire two data sets from a system, rather, you can partition one data set into two using the SI Split Signals VI. The SI Split Signals VI divides the preprocessed data samples into a portion for model estimation and a portion for model validation.

In Figure 9-10, the **1st portion (%)** is 66, which means the SI Estimate Orders of System Model VI will use 66% of the data samples for estimation and the remainder of the data samples for validation.

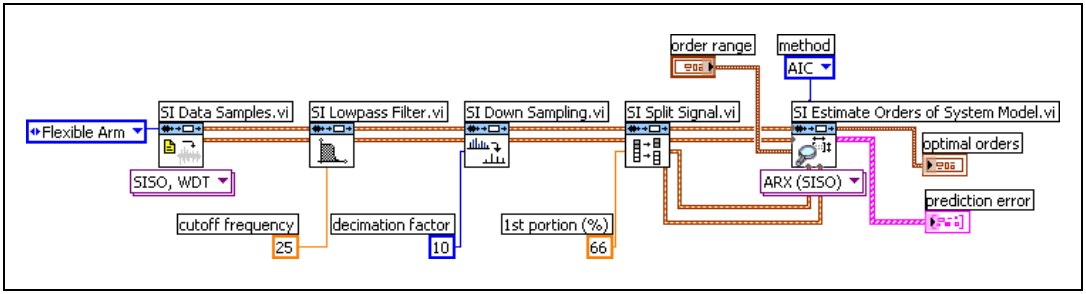

Figure 9-10. Estimate Orders of ARX Model VI

The SI Estimate Orders of System Model VI generates the **prediction error** plot for the ARX model and the optimal A order and B order based on the AIC criterion. By using the AIC criterion, the lowest prediction error corresponds to a model dimension of 19, as shown in Figure 9-9. For an ARX model, the model dimension is equal to the sum of the A order, B order, and delay values. The SI Estimate Orders of System Model VI returns the following optimal orders:

- A order = 9
- B order = 10
- delay = 0

Verifying the Results

After determining the orders of the model, you want to verify the results to ensure the model accurately describes the system. One method is to plot a pole-zero map and visually inspect the plot to determine whether there is any redundancy in the data. If a pole and a zero overlap, the pole and zero cancel out each other, which indicates the estimated optimal order is too high.

The **pole-zero plot** graph in Figure 9-11 shows a pole-zero plot with three overlapping pole-zero pairs. Due to numerical error, it is unlikely that a zero and a pole perfectly overlap. You can use the confidence region to justify whether the pole and the zero cancel out each other.

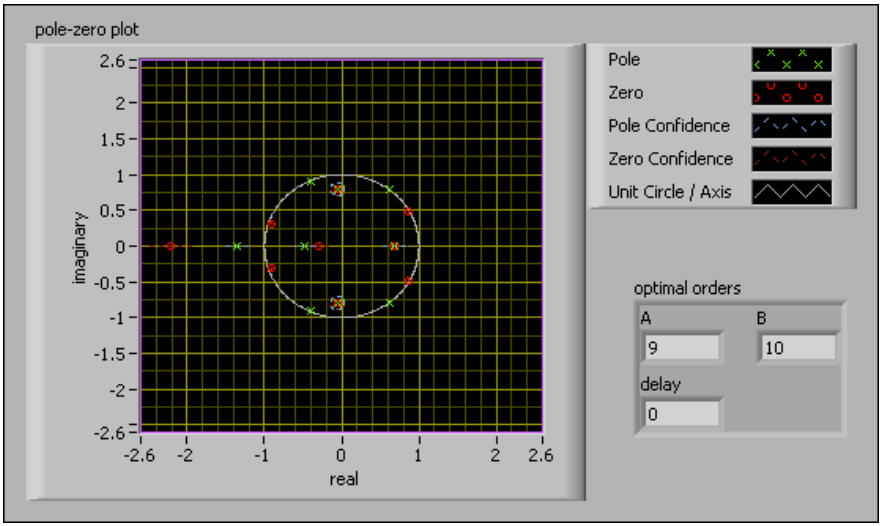

Figure 9-11. Pole-Zero Plot for an ARX Model

Because there are three pole-zero pairs, you can conclude that the AIC criterion does not produce the most optimal orders.

Minimum Description Length Criterion

Because the AIC criterion produced a model with non-optimal orders, you can try estimating the model orders with the MDL criterion. By using the MDL criterion, the lowest prediction error corresponds to a model dimension of 12, as shown in Figure 9-9. The SI Estimate Orders of System Model VI returns the following optimal orders:

- A order = 6
- B order = 6
- delay = 0

Figure 9-12 shows a pole-zero plot of a model with a model dimension of 12.

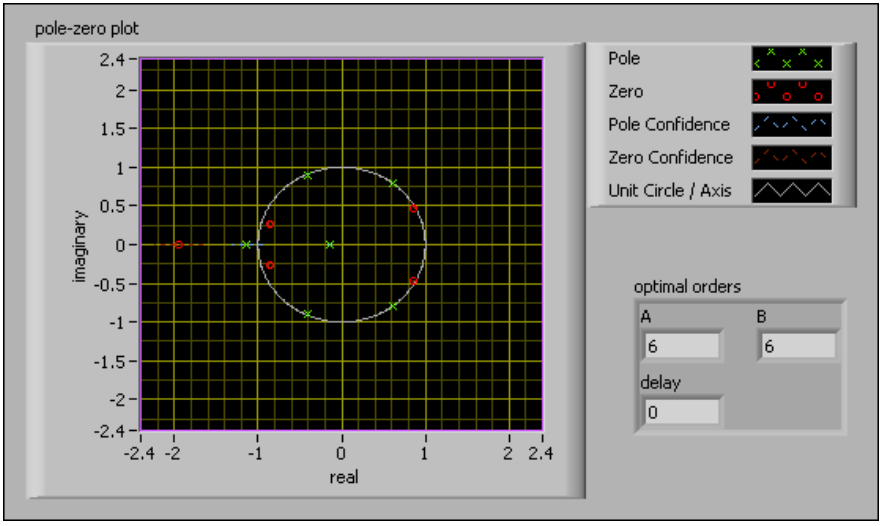

Figure 9-12. Pole-Zero Plot for a MDL Model

Compare Figure 9-12, which uses the MDL criterion and Figure 9-11, which uses the AIC criterion. Because there are no overlapping pole-zero pairs in Figure 9-12, you can conclude that the MDL criterion fits better than the AIC criterion in this particular example.

In addition to examining redundancy, you also can use the pole-zero plot for other purposes. For example, both Figure 9-11 and Figure 9-12 show poles outside the unit circle. Having poles outside the unit circle implies that this model is not optimal because the ARX system based on the AIC or MDL criteria is unstable. One way to stabilize the system is to change the order.

In addition to the FPE, AIC, and MDL criteria, you can set user-defined orders in the SI Estimate Orders of System Model VI.

User-Defined Criterion

If you know nothing about the system, you might have to rely on trial and error to determine the optimal orders of the model. However, if you have some knowledge about a system, you can customize the estimation to find a model that fits a certain model dimension. For this model, assume you know that the system is stable; therefore, no poles exist outside the unit circle. Because both the AIC and MDL criterion did not produce stable models, the model orders do not describe the system accurately.

On the block diagram, as shown in Figure 9-13, you can customize the **method** parameter. Instead of **AIC** or **MDL**, you can select **<Other>** and enter the desired model dimension in the textbox. Assume you know that the model dimension is nine.

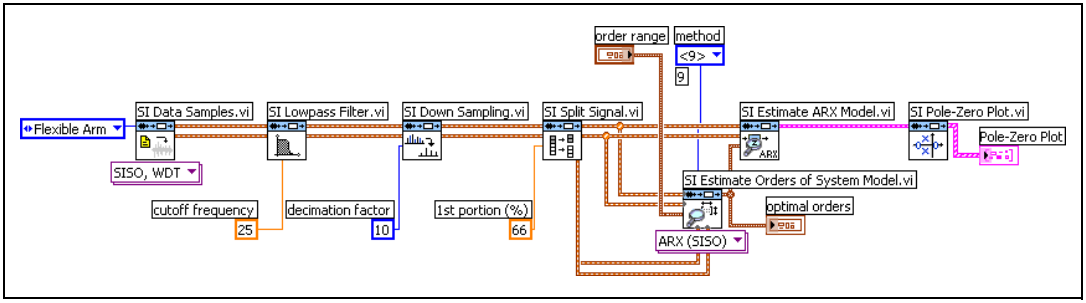

Figure 9-13. Pole-Zero Cancellation VI

Figure 9-14 shows the corresponding **pole-zero plot** graph with a model dimension of nine, which corresponds to the following optimal orders:

- A order = 4
- B order = 5
- delay = 0

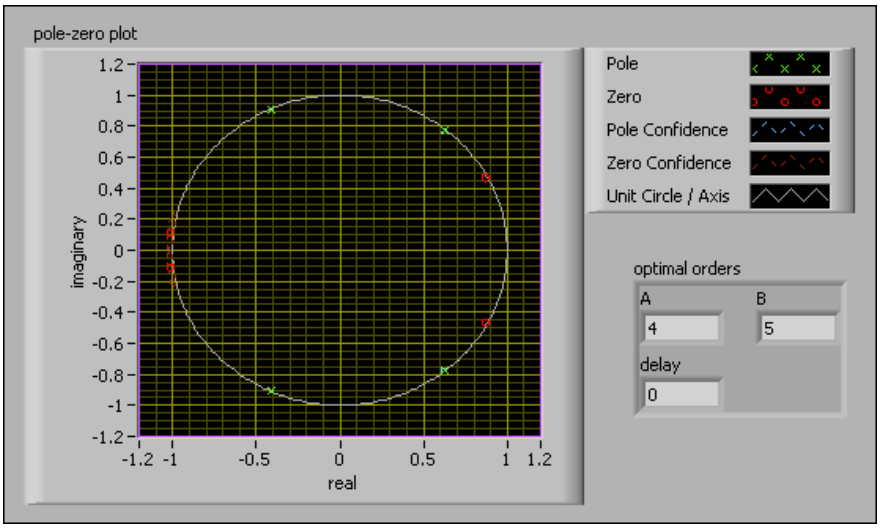

Figure 9-14. Pole-Zero Plot with User-Defined Orders

Compare the pole-zero plot in Figure 9-14 with Figure 9-11 and Figure 9-12. Figure 9-14 has no overlapping pole-zero pairs and all the poles are within the unit circle. By visually inspecting the pole-zero plot, you can see that this model is stable and not redundant. Using these model orders, you now can estimate and verify the system model.

ARX Model Validation

The goal of model validation is to determine whether or not the estimated model accurately reflects the actual system. Using the model orders found in the *User-Defined Criterion* section, you can simulate and predict the response of the system. You can compare these responses to the actual response and determine the accuracy of the estimated model. You also can analyze the residuals to determine the accuracy of the estimated model. Refer to the *Validating Models* section of Chapter 8, *Analyzing, Validating, and Converting Models*, for more information about validating a model.

The following sections describe how to apply these techniques to model validation.

Simulation and Prediction

You can use the SI Model Simulation VI and SI Model Prediction VI to determine the accuracy of the estimated model. The SI Model Simulation VI simulates the system model and the SI Model Prediction VI performs a prediction of the system model. The results of the SI Model Prediction VI might differ from the SI Model Simulation VI because the SI Model Prediction VI periodically makes corrections to the estimated response based on the actual response of the system.

Figure 9-15 shows how you use these VIs to verify the ARX model created in the *User-Defined Criterion* section.

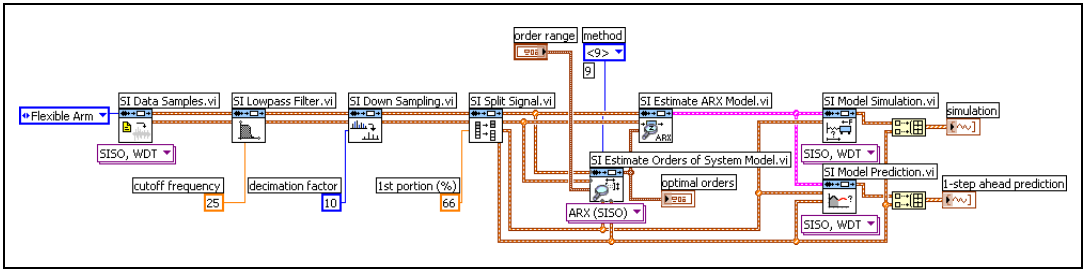

Figure 9-15. Simulation & Prediction with ARX Model VI

The **simulation** and **1-step ahead prediction** graphs enable you to visually determine how accurate the model is. Figure 9-16 shows the results of the simulation and prediction as well as the actual response of the system.

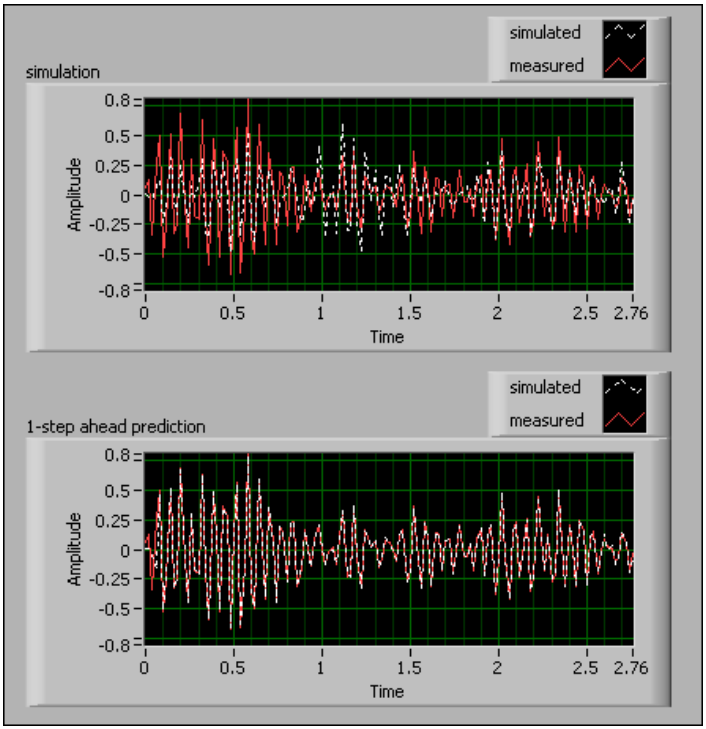

Figure 9-16. Simulation and Prediction Graphs for an ARX Model

Notice how the actual response, or the **measured response**, is different from the **simulated response** in the **simulation** graph. The SI Model Simulation VI simulates the response of the system without considering the actual response of and the noise dynamics in the system.

Residual Analysis

In addition to simulation and prediction, you can perform a residual analysis to validate the system model. Residual analysis tests whether the prediction error correlates to the stimulus signal. Prediction errors are usually uncorrelated with all stimulus signals in an open-loop system.

The block diagram in Figure 9-17 shows how you can use the SI Model Residual Analysis VI with the ARX model identified in the *User-Defined Criterion* section to analyze the residuals.

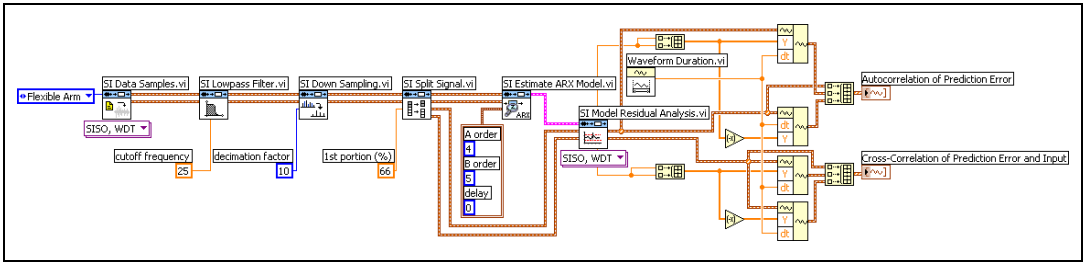

Figure 9-17. Residual Analysis VI

Figure 9-18 shows an example of ideal results where both autocorrelation and cross correlation are inside the confidence region except those in the vicinity of $\tau = 0$. This result indicates that the estimated model accurately describes the system.

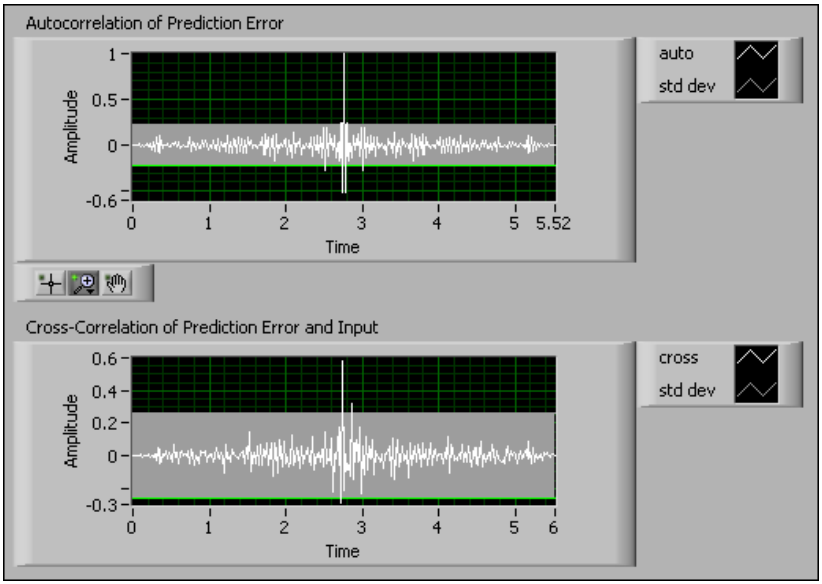

Figure 9-18. Residual Analysis for A Order = 4, B Order = 5, and Delay = 0

When you verify and validate the identified model, you must use multiple analysis techniques to determine if the estimated model accurately represents the system. Some analysis techniques can be misleading. For example, if you performed a residual analysis on the model identified in the *Minimum Description Length Criterion* section, you might conclude that this model is an accurate representation of the system. Figure 9-19 shows the autocorrelation and cross correlation residual analysis for the model in

the *Minimum Description Length Criterion* section. Recall that this model has the following orders:

- A order = 6
- B order = 6
- delay = 0

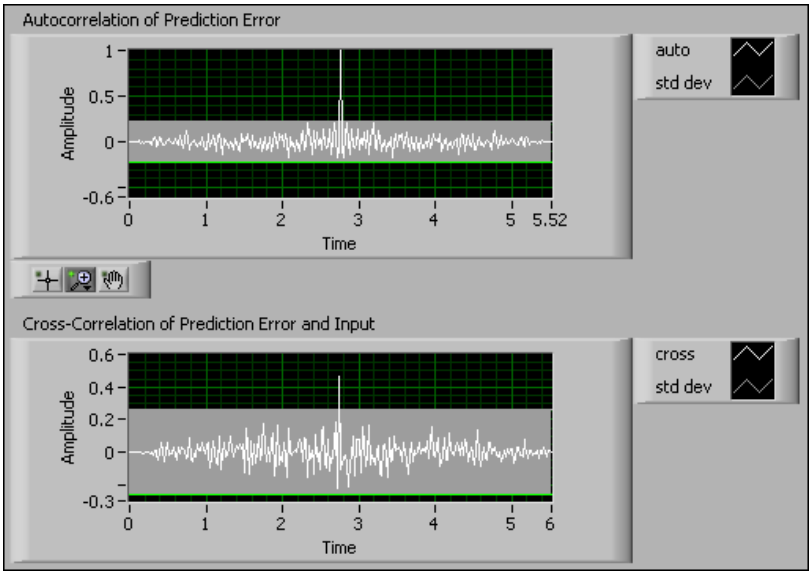

Figure 9-19. Residual Analysis of ARX Model with A Order = 6, B Order = 6, and Delay = 0

Figure 9-19 shows that both the autocorrelation and cross correlation are inside the confidence region. Therefore, without performing any other analyses, you might conclude that this model is an accurate representation of the system. However, the pole-zero analysis in the *Minimum Description Length Criterion* section showed poles outside of the unit circle. So you already determined that this model is unstable. Thus, despite acceptable autocorrelation and cross correlation values, concluding that this model is accurate is incorrect.

Thus, if you only performed a residual analysis, you might not discover that this model is actually unstable. When validating a model, perform multiple analyses to ensure the accuracy of the model.

Estimating a State-Space Model

For a state-space model, order estimation is equivalent to estimating the number of significant singular values, which correspond to the number of states in the model. After identifying a state-space model that represents the system, you can use the same validation and verification technique used in the *Simulation and Prediction* section and the *Residual Analysis* section.

Refer to the *State-Space Model* section of Chapter 4, *Parametric Model Estimation Methods*, for more information about estimating state-space models.

The examples in this section use the same flexible robotic arm data and the same preprocessing techniques.

Finding the Singular Values

The block diagram in Figure 9-20 shows how to use the SI Estimate Orders of System Model VI to find the optimal order and the number of significant singular values.

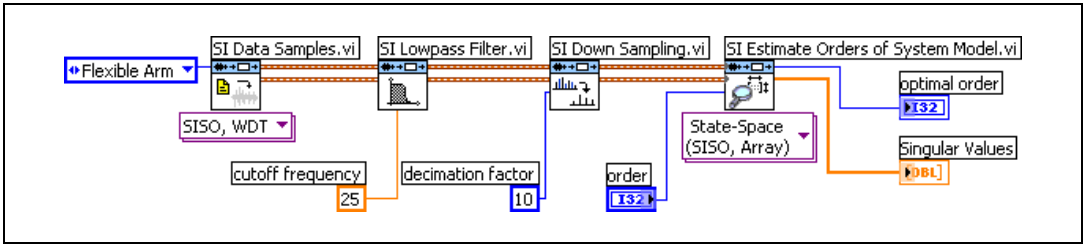

Figure 9-20. Estimate Orders of State-Space Model VI

The **Singular Values** graph in Figure 9-21 shows a singular value plot with four leading singular values.

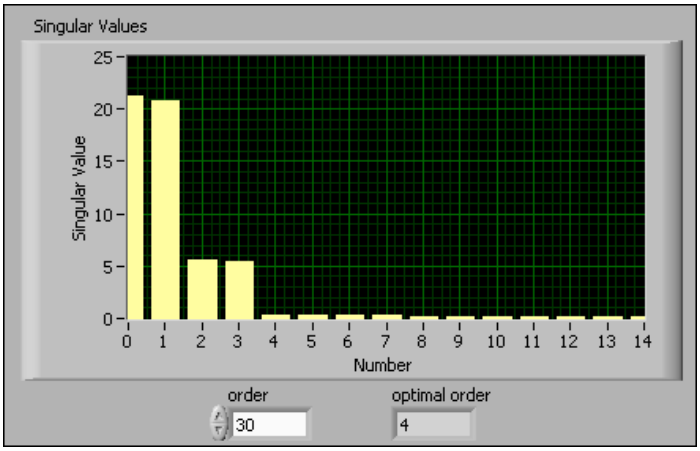

Figure 9-21. Singular Value Plot for State-Space Model

By looking both at the **Singular Values** graph and the **optimal order**, you can see that there are four states in this state-space model.

Validating the Estimated State-Space Model

You can validate the state-space model in the same way that you validated the ARX model. You use the SI Model Simulation VI and the SI Model Prediction VI to determine the accuracy of the state-space model.

Figure 9-22 shows the complete process, from estimating the state-space model to simulating and predicting the response of the model.

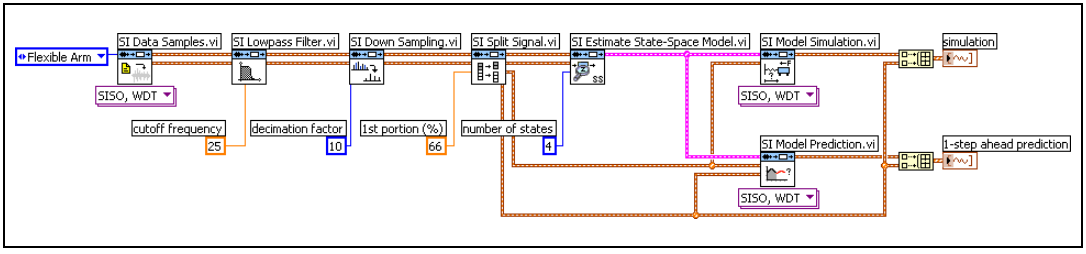

Figure 9-22. Simulation & Prediction with State-Space Model VI

The **simulation** and **1-step ahead prediction** graphs in Figure 9-22 show simulation and prediction plots for a state-space model.

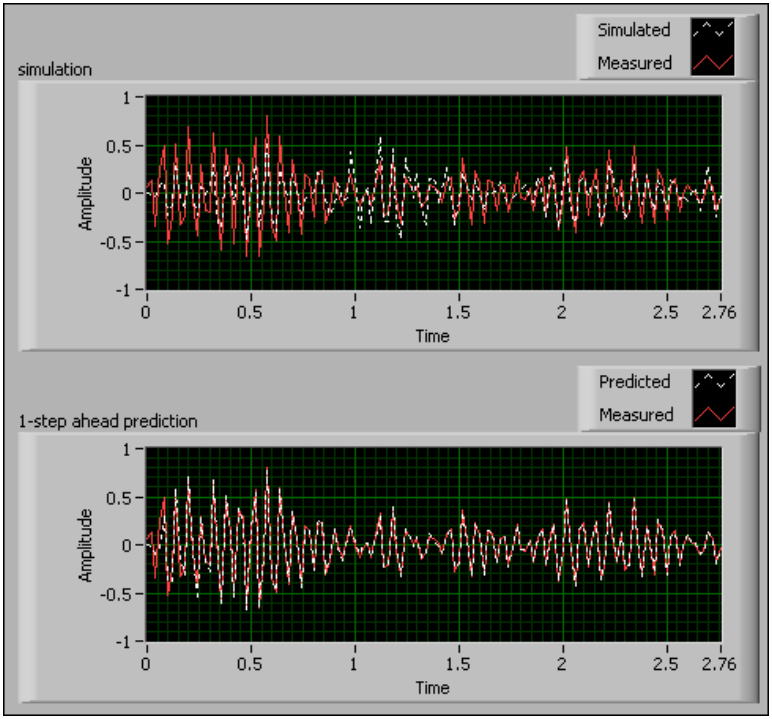

Figure 9-23. Simulation and Prediction Graphs for a State-Space Model

Refer to the *Validating Models* section of Chapter 8, *Analyzing, Validating, and Converting Models*, for more information about validating a model.

Additional Examples

In this chapter, you learned how to start from raw data and find an accurate model to represent the system from which you acquired the data. The examples in this chapter are located in the `labview\examples\system identification\SICaseStudy1.llb`.

The `labview\examples\system identification\` directory also contains other examples you can use to become familiar with the System Identification VIs. You also can access the System Identification example VIs by selecting **Help»Find Examples** to display the NI Example Finder and then navigating to the **Toolkits and Modules»System Identification** folder.

Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for more information about specific System Identification VIs in this case study.

A

References

This manual contains information about using the LabVIEW System Identification Toolkit VIs in LabVIEW applications. In most cases, the underlying theory is omitted. This list of references contains more information about the theory and algorithms implemented in the System Identification Toolkit.

Astrom, K. J., and B. Wittenmark. 1974. *Adaptive control*. 2nd ed. Addison-Wesley.

De Moor, B. 1988. *Mathematical concepts and techniques for modelling of static and dynamic systems*. Ph.D. thesis. Katholieke Universiteit Leuven, UDC 519.17.

Gill, P. E., W. Murray, and M. H. Wright. 1981. *Practical optimization*. Academic Press.

Goodwin, G. C., and R. L. Payne. 1977. *Dynamic system identification: Experiment design and data analysis*. Academic Press.

Kailath, T. 1980. *Linear systems*. Prentice Hall.

Ljung, L. 1999. *System Identification Theory for the User.* 2nd ed. Prentice Hall.

Ljung, L., and T. Glad. 1994. *Modeling of dynamic systems*. Prentice Hall.

Ljung, L., and T. Söderström. 1983. *Theory and practice of recursive identification*. MIT Press.

Oppenheim, A. V., and R. W. Schafer. 1989. *Discrete-time signal processing*. 2nd ed. Prentice Hall.

Oppenheim, A. V., A. S. Willsky, and I. T. Young. 1997. *Signals and systems*. Prentice Hall.

Overschee, P. V., and B. De Moor. 1993. *N4SID: Subspace algorithms for the stochastic identification problem*. Automatica 29, no. 3:649–660.

———. 1994. *Subspace algorithms for the identification of combined deterministic-stochastic systems*. Automatica 30, no. 1:75-93.

Powell, M. J. D. 1964. *An efficient method for finding the minimum of a function of several variables without calculating derivatives*. Computer Journal 7:155–162.

Priestley, M. 1981. *Spectral analysis and time series*. Academic Press.

B

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, go to ni.com/services or contact your local branch at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.